

THE WEB DESIGN BOOK

The ultimate guide to creating stunning modern websites



THE WEB DESIGN BOOK

The world of web design changes on a daily basis. Whether it's a new language that is being invented, or a new design element, or even just a new trend or fashion. Gone are the days where coding was just about a page on the web, these days this evolved, responsive way of creating that goes as far as building games and applications for your mobile devices. In this book we aim to take you through a plethora of different ways to make your website cutting edge and brief you on the new standard of JavaScript. You will find all the tutorial files you need over on FileSilo along with over 10 hours of video tutorials. Enjoy the book!

THE WEB DESIGN BOOK

Imagine Publishing Ltd
Richmond House
33 Richmond Hill
Bournemouth
Dorset BH2 6EZ
☎ +44 (0) 1202 586200

Website: www.imagine-publishing.co.uk

Twitter: @Books_Imagine

Facebook: www.facebook.com/ImagineBookazines

Publishing Director
Aaron Asadi

Head of Design
Ross Andrews

Production Editor
Jen Neal

Senior Art Editor
Greg Whitaker

Designer
Abbi Denney

Printed by
William Gibbons, 26 Planetary Road, Willenhall, West Midlands, WV13 3XT

Distributed in the UK, Eire & the Rest of the World by:
Marketforce, Blue Fin Building, 110 Southwark Street, London, SE1 0SU
Tel 0203 148 3300 www.marketforce.co.uk

Distributed in Australia by:
Network Services (a division of Bauer Media Group), Level 21 Civic Tower, 66-68 Goulburn Street,
Sydney, New South Wales 2000, Australia Tel +61 2 8667 5288

Disclaimer
The publisher cannot accept responsibility for any unsolicited material lost or damaged in the post. All text and layout is the copyright of Imagine Publishing Ltd. Nothing in this bookazine may be reproduced in whole or part without the written permission of the publisher. All copyrights are recognised and used specifically for the purpose of criticism and review. Although the bookazine has endeavoured to ensure all information is correct at time of print, prices and availability may change. This bookazine is fully independent and not affiliated in any way with the companies mentioned herein.

The Web Design Book Volume 5 © 2015 Imagine Publishing Ltd

ISBN 978-1785461101

Part of the
**web
designer**
bookazine series



Design like a pro, code like an expert

8 Create cutting edge designs

Google Analytics

16 Use Photoshop to complement your design

HTML & CSS

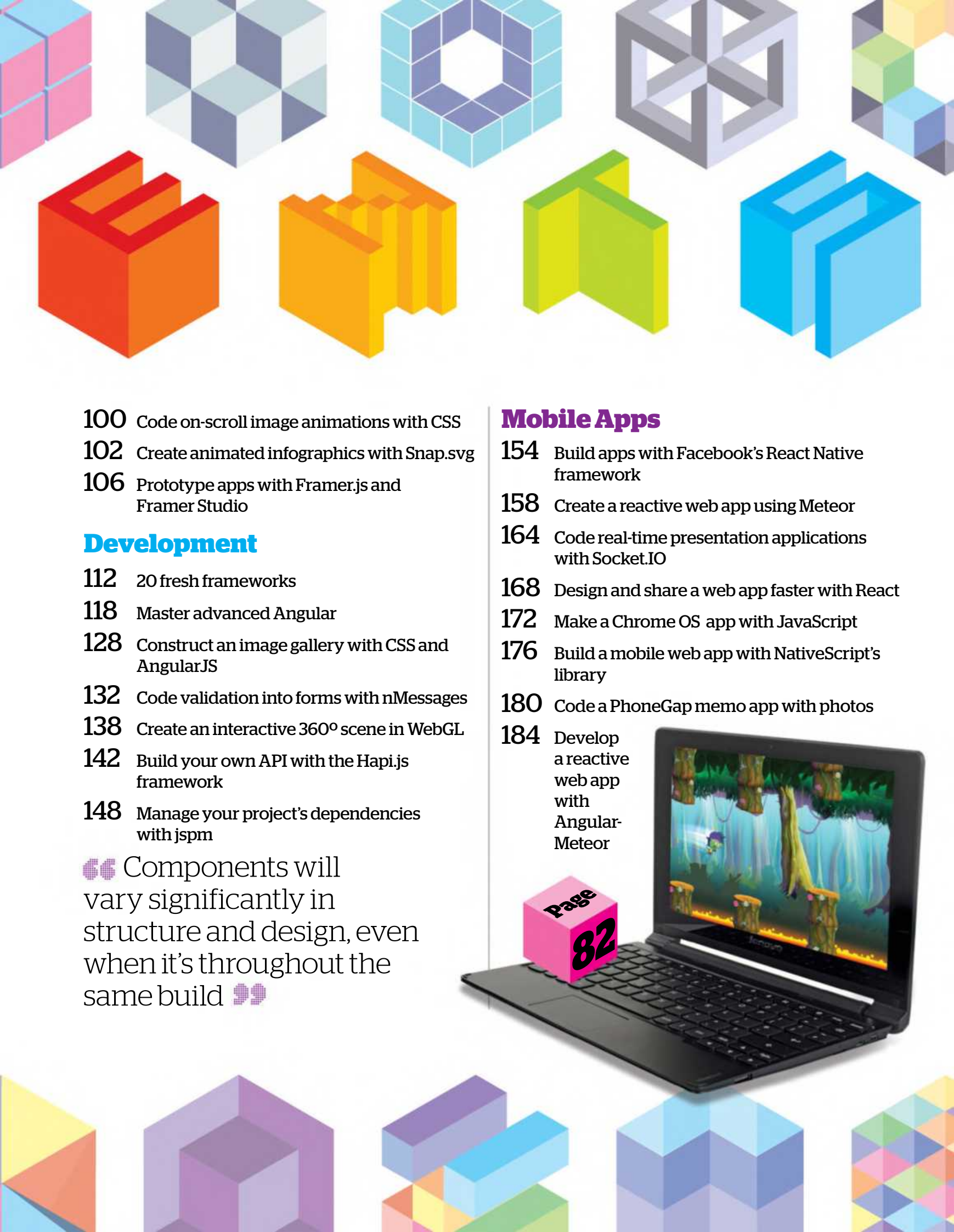
- 24 Power up CSS preprocessors
- 32 Set up your own grid system using CSS
- 36 Make pop-up modal boxes with pure CSS
- 40 Build a responsive fixed page border
- 42 Design aspect ratio based layouts
- 46 Generate sweeping animations with CSS
- 50 Make a screen shrink on scroll
- 52 Produce an animated off-screen 3D menu
- 56 Slide up titles on page load using CSS
- 58 Animate scroll-reveal split backgrounds
- 60 Animate an information card box with CSS
- 64 Construct an adaptive printable CSS design

Page
8



jQuery & JavaScript

- 70 The new JavaScript standard
- 76 jQuery special effects
- 82 Make dynamic graphics with the p5.js library
- 86 Create shuffling text effects with jQuery
- 88 Design a 2D game using the Pixi engine
- 92 Sync animations to audio and video with Popcorn.js
- 96 Animate SVGs with the Vivus.js library

- 
- 100 Code on-scroll image animations with CSS
 - 102 Create animated infographics with Snap.svg
 - 106 Prototype apps with Framer.js and Framer Studio

Development

- 112 20 fresh frameworks
- 118 Master advanced Angular
- 128 Construct an image gallery with CSS and AngularJS
- 132 Code validation into forms with nMessages
- 138 Create an interactive 360° scene in WebGL
- 142 Build your own API with the Hapi.js framework
- 148 Manage your project's dependencies with jspm

☞ Components will vary significantly in structure and design, even when it's throughout the same build ☞

Mobile Apps

- 154 Build apps with Facebook's React Native framework
- 158 Create a reactive web app using Meteor
- 164 Code real-time presentation applications with Socket.IO
- 168 Design and share a web app faster with React
- 172 Make a Chrome OS app with JavaScript
- 176 Build a mobile web app with NativeScript's library
- 180 Code a PhoneGap memo app with photos
- 184 Develop a reactive web app with Angular-Meteor



Code like an expert



DOWNLOAD TUTORIAL FILES

www.filesilo.co.uk/bks-747



DESIGN

{ **LIKE A PRO** }

CODE

< **LIKE AN EXPERT** >

**Get your hands on a host of HTML5 tips, trick
and techniques to empower your frontend
and backend**

Using <article> for semantic components

THE <ARTICLE> TAG IS ONE OF THE MOST MISUNDERSTOOD ELEMENTS OF HTML5

Marking up widgets and components can be tricky to get right semantically. Components will vary significantly in structure and design, even when it's throughout the same build.

Unfortunately, this is at odds with what we actually want as developers: a consistent maintainable approach with semantic value. Well this is where the <article> tag comes into play.

"The article element represents a complete, or self-contained, composition in a [...] site [...] that is, in principle, independently distributable or reuseable, e.g. in syndication. This could be [...] a magazine or newspaper article, [...] an interactive widget, [...] or any other independent item of content" as defined by the HTML5 specification at w3.org/TR/html5/sections.html#the-article-element.

The specification makes it very clear that the <article> tag has a much wider use case than is first obvious. This makes it the perfect semantic tool for components, which we often design in modular fashions.

The HTML5 specification also details that the <article> tag's content model is flow content, which is the way we mark up most of the elements that are used in the body of documents. This means that the <article> tag becomes the semantic choice when we want to contain modular content and even functionality such as things like forms and navigation.

Of particular note is that the <article> tag can contain the new HTML5 elements header, footer, aside and more instances of itself - allowing for further semantic structure inside your reusable component. This provides us with semantic building blocks for complex modular UIs such as faceted search, accordion menus and tab systems - perhaps even all three at once (though this is not advised).

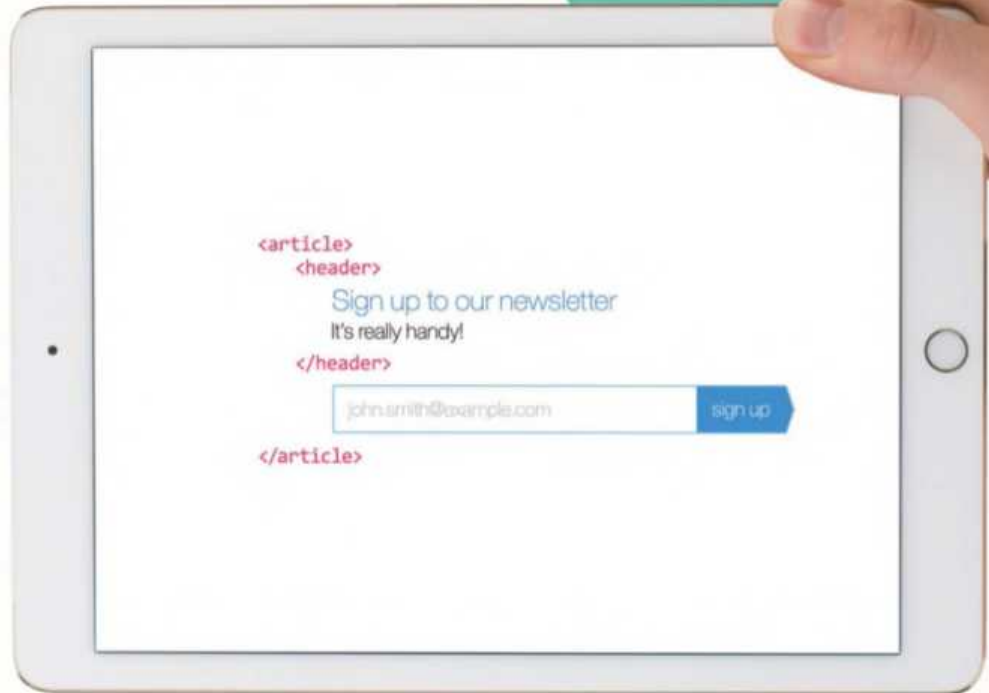
In many ways <article> is a 'do everything' tag. It's important to remember that the <article> tag brings semantic value only to components that can be isolated from the rest of the page.

It is not the right tag for supporting 'pods' or other components that only make sense in situ or are otherwise relying on surrounding content.

Likewise, the <article> tag is very unlikely to be the right tag in cases where the component is semirelated but also independent, for example in a pull quote. In these scenarios then, the <aside> tag is likely the best choice. Though it's worth considering that aside elements can contain article elements - enabling the semantic markup of any tangentially related content which also containing reusable components.

AUTOCOMPLETE TURNED OFF

Most browsers now have an autocomplete defaulted to 'on' for <input>. Turn this off for things like reset password fields.



“The <article> tag becomes the semantic choice when we want to contain modular content and even functionality”



“It’s a little detail a lot of websites miss, but form fields should only autocomplete information that is likely to be correct, lest users spend time deleting text that shouldn’t be there in the first place.”

Ralph Saunders
Front-end developer
at Redweb

Misunderstood HTML5 tags

SOME HTML5 ELEMENTS AREN'T WHAT THEY FIRST SEEM

Section

The `<section>` tag provides a semantic way to markup thematically grouped content that should be in the document outline (so it should have a heading). The `<section>` element would be perfect for marking up chapters in a book for example. We can also make use of it inside flow content, where we enable the clear separation of many sections in a lengthy article – such as a thesis or dissertation.

Footer

The `<footer>` tag is not for website footers but for footer content that relate to the nearest sectioning element (these are the four sectioning elements: section, article, aside and nav). The footer can also be used for content such as that of bylines, related documents and copyright data. It can also contain entire sections and this will then lend itself to things like appendices, indexes and licence agreements.

Header

Like the `<footer>` tag, the `<header>` tag isn't for website headers – as you would be forgiven for thinking. It's for introductory content relating to the nearest sectioning element. It can be used for larger-scale content such as a table of contents or foreword, but is more commonly used for headings and standfirst paragraphs.

Address

The `<address>` tag is not just for addresses you can post letters to. It is also contact information for the nearest `<article>` or even `<body>` when it's applicable to the document as a whole. While this may be a postal address sometimes, it's more likely an email address, phone number or link to the author's page.

Nav

The `<nav>` tag can be used for primary navigation, but it should also be used to markup any navigation on the page. Secondary navigation, even when internal to the page, should be marked up with `<nav>` – like jump links. If the nav contents represent a list of items they should be marked up with a list as well.

HTML5 CHEAT SHEET

There are a lot of tags in HTML5, some popular and some not so well known. A quick reference to a host of HTML5 tags is just the job. Get your hands on one at websitesetup.org/html5-cheat-sheet.



ARIA landmarks

USE ARIA LANDMARKS TO IDENTIFY REGIONS OF A PAGE

ARIA stands for Accessible Rich Internet Applications and is a set of attributes for making markup more accessible. The basic premise is that these attributes can help describe the components on a page for screen reader applications. A subset of these attributes are called landmark roles, and they indicate navigational landmarks like in this code example:

```
<article role="main">
  <header>
    <h1>Awh, kitteh</h1>
    <p>We've just named our new cat
    Jimmy!</p>
  </header>
</article>
```

Here we're declaring this section of the page the 'main' content of the document – it contains the content you came to the page for. There are a few more landmarks that should go into any web project.

'Banner' declares that the region is mostly site oriented as opposed to page oriented. This can be applied to site headers where the logo, primary navigation and search are site specific.

'Complementary' declares a region that's outside the main content but complements it and would remain meaningful in isolation. Most sidebars fit this criteria.

'Navigation' declares the region contains navigational elements for navigating the page or related pages. If you're using `<nav>` you should be using `role="navigation"`.

Optimise page structure for Time To First Paint

OPTIMISE FOR TTFP TO GET MORE PAGE VIEWS AND IMPROVED CONVERSIONS

TTFP is the metric for measuring how fast the browser was able to paint the page. We

want to minimise the time a user spends looking at a white screen by optimising assets and the order in which they're loaded.

When visiting a page the browser must download the HTML of that page before it fetches the CSS and JavaScript assets linked within it. There are a couple of things we can do to ensure only the minimum number of assets are fetched before the browser paints. Most people know that JavaScript should go at the bottom of a page just before the `</body>` tag. What's less known is

that if your JavaScript isn't interacting with the CSSOM you can include an 'async' attribute to turn it into a nonblocking request, meaning it won't hinder TTFP.

```
<script src="app.js" async></script>
```

We can also use the media attribute in links to CSS assets to make them nonblocking.

```
<link href="print.css" rel="stylesheet"
media="print" />
<link href="desktop.css" rel="stylesheet"
media="(min-width: 1024px)" />
```

Desktop.css won't delay TTFP for devices that don't meet the requirements of the media query, it gets loaded later.



HTML5 input types for better keyboards

BRINGING NATIVE INPUT VALIDATION, AND UX GOODIES TO THE TABLE

Mobile devices often have small keyboards where data entry is made all the harder. We can ease our users' pain by using the new input types that come with HTML5, which can change a keyboard layout intuitively on some mobile platforms out there.

While using these types correctly won't fix a bad form, the different layouts go a long way in making forms less stressful to use on small devices. Some of the more common input types you can include in your forms straightaway are below:

`<input type="tel" name="telephone">`

Tel inputs will bring up a numeric keyboard. No more pesky small keys when entering phone numbers. No format validation is enforced with this type.

`<input type="number" name="amount of pets">`

Number inputs will also bring up a numeric keyboard. Only numbers can be put into this input.

`<input type="email" name="email">`

Email inputs will bring up a keyboard with an @ sign.

Browsers will do email address validation on this input type.

`<input type="url" name="website">`

Url inputs will bring . / and .com buttons to the interface. The browser will validate that this looks like a URL.

`<input type="date" name="date of visit">`

Date inputs will bring up a date picker on modern desktop browsers and a scrolling interface on most mobile devices.

“Different layouts go a long way in making forms less stressful”

Use rel="prefetch" to enhance browsing experience

PREFETCH CAN SPEED UP BROWSING EFFECTIVELY

HTML5 prefetching uses the rel attribute on <link> tags and looks like this:

```
<link rel="prefetch" href="/Page2.html" />
```

Page2.html will be fetched by the browser when nothing else is happening and it's fetched before the user decides to go there. Server response time is often the largest delay in fetching new pages and other assets will be cached, the savings here are substantial.

URL prefetching is a double-edged sword. We're downloading a resource that the user may not ever navigate to and on a mobile network it'll count against your user's data allowance. With this in mind, we should only prefetch resources we're pretty sure the user will visit. There are some common resources that are always worth prefetching:

1. The next page in paginated search results
2. The next page in sequential multipage forms
3. Pages with only one major call to action that the user will likely visit eventually
4. Common images on most pages, like spritesheets.

Along with standard prefetching there's also DNS prefetching which works in a similar fashion.



Darren Hickling

Web engineer at Redweb
“HTML5 introduced new input types, but also new attributes for them. Alongside pattern, these new input types have proven very useful in recent projects. In supported browsers, they can also remove the need for custom JavaScript validation and are enforced by the browser so they cannot be ignored.”



Phil Heywood

Creative director at Redweb
“Web pages using dynamic content and user interface components used to cause problems for people with certain accessibility needs.

HTML5, in conjunction with WAI-ARIA, makes it possible for everybody to use rich Internet applications, even if they rely on screen readers and other assistive technologies.”



Ryan Kerry

Global head of development at Lush

“HTML5 introduced the contentEditable attribute, which enables users the ability to alter the contents of a HTML page. This has proven to be extremely useful when required to perform rapid in-browser prototyping of web pages that require real-time updates to the CSS output for clients to see.”



Jonathan Ginn

Developer evangelist at We Are Base

“One of my favourite underutilised new HTML elements is the datalist.

It's great for letting someone perform a search while offering find-as-you-type options, all in HTML5 with no JavaScript. It's not fully supported yet but like all HTML5 elements, it has graceful fallbacks for older browsers.”

Back-end APIs for apps

USE HTML5 TO MAKE APPS WITHOUT NATIVE PLUGINS

WebRTC

Don't be fooled, WebRTC has nothing to do with real-time clocks. Instead, brace yourself for a standards-based way to perform voice and video calls using nothing but a web browser. When sufficient traction has been gained, Web Real Time Communication might swipe away Skype, MSN Messenger and other proprietary protocols. Sadly, its adoption has been slow so far – it is limited to Chrome, Opera and Firefox only.

WebWorkers

JavaScript and multithreading are like cat and water: when the language was first designed, no one thought of multicore processors.

WebWorkers seek to remedy this design deficiency by introducing a special and convenient way to run JavaScript logic in a separate thread. A message-passing protocol is then used for data exchange between the worker and the main app.

Shadow DOM

Even though Web Components have not been formalised as of this writing, the shadow DOM feature already deserves attention. It permits the isolation of subcomponents from the rest of the page. When stored in a ShadowRoot instance, your widget's elements are shielded from well-meaning but badly written enumeration routines seeking to wreak havoc on their finely tuned composition.

Typed arrays

If JavaScript wants to become a full-featured programming language, a way must be found to enable interaction with native libraries. Sadly, its untyped memory model makes the management of memory buffers difficult. In essence, a typed array is little more than a broker between a slab of memory and JavaScript's native types. You can use them to create arrays of floats or integers though, and these can then be passed onto any native functions.

File API

HTML5 introduces a plethora of storage systems. Sadly, sometimes there is just no alternative to a good old file. The File API permits you to access pesky little and chunky large files sitting in the file system of your client's workstation. It comes complete with a set of widgets to make selecting the files easy.

Where are we?

THE GEOLOCATION API HELPS YOU FIGURE OUT THE USER'S LOCATION



1. Where is our user?

Even though the HTML5 Geolocation API has seen some impressive implementations, some browsers still don't implement the library correctly. The first step to a working GIS application involves the querying of the presence of the geolocation object. If found, the location getter methods can then be invoked. GetCurrentPosition takes one, two or three parameters: one method which will be called on success, an optional second one which will be called on failure and an optional JSON object with further parameters to set up accuracy and data source selection.

2. Handling all errors!

Getting a user's location is all but foolproof. The optional error handling method receives an object containing four different error codes, eg PERMISSION_DENIED is returned if the user denies the location request shown as the website tries to access the Geolocation API for the first time. Note that some browsers don't return an error if the user refuses to divulge to your application. On such browsers, the success callback simply doesn't get invoked – one way to handle this behaviour is with setTimeout. It can spin a checker to inspect whether success and/or failure have invoked.

3. Parsing user data

If the locationeering process was successful, you are provided with a position object. It contains the set of fields shown in the following table:

Field	Data
accuracy	Accuracy of Lat and Lon data
altitude	Altitude from mean sea level
altitudeAccuracy	Accuracy of altitude
heading	Heading, degrees from north
latitude	Latitude
longitude	Longitude
speed	Movement speed
timestamp	Age of data

4. Caveat emptor

Keep in mind that the actual implementations differ on legacy browsers. For example, Internet Explorer did not support the official Geolocation API until IE9. Fortunately, an intermediary broker layer has been developed. It acts as a relay station between your code and a variety of proprietary implementations. Further information on it can be discovered by visiting the website of the project (github.com/estebanav/javascript-mobile-desktop-geolocation). Another source of grief involves the accuracy of the data: most desktops are located via their IP address. This can lead to significant accuracy issues.

10 HTML5 tips

MAKE THE MOST OF YOUR BACKEND WITH THESE TRICKS

HTML Imports are dead

During the specification of Web Components, Mozilla came up with an include directive for HTML files:

```
<link rel="import" href="myfile.html">
```

Sadly, they never managed to pick up any form of significant traction. Since Firefox dropped support for them some time ago, it's better to leave them alone.

Force some input

HTML5 saves validation effort by informing the browser that some fields must be filled:

```
Name: <input type="text"
name="mandatoryname" required>
```

By default, most browsers will display an exclamation mark in the field as long as it is empty.

Force a pattern

For some types of data, a predefined format is very helpful for us as it can be enforced in order for us to weed out any invalid input from our code. Take a look at our snippet below because it shows how HTML5 can be harnessed in this instance to ensure that our users don't end up accidentally inputting non-IPv4-formatted addresses:

```
<input type="text" pattern="\
d{1,3}\.\d{1,3}\.\
d{1,3}\.\d{1,3}">
```

Annotate data

HTML always found itself in a pressure field between verbosity and compactness: adding new tags makes the language richer, but increases complexity.

Microdata solves this by specifying an external format for metadata. The markup is then enriched with tags referencing the metadata stylesheet:

```
<section itemscope
itemtype="http://data-
vocabulary.org/Person">
<h1 itemprop="name">Mark
Pilgrim</h1>
```

Specify your own

All kinds of actors have created a large selection of metadata stylesheets, which can be embedded into your designs easily. But of course, a custom stylesheet is needed in some sticky cases. More information (as well as a large range of information on the microdata data model, markups and snippets) and these can be found at the website of Mark Pilgrim, which is available at diveintohtml5.info/extensibility.html.

Using gauges

Digital aeronautical systems have always suffered from a weakness of the human brain: textual information is more difficult to parse. Fortunately, Gauges can be created on HTML5-capable browsers by using the following, widely supported tag:

```
<meter value="6" min="0" max="10">6 of 10</
meter><br>
```

<Progress> and <meter>

Even though there is nothing to stop you from using a Gauge as a progress bar, HTML5 provides a dedicated option. Its use is demonstrated in the following snippet - keep in mind, that some versions of Internet Explorer don't support the <progress> and <meter> tags:

```
<progress value="66" max="100">
</progress>
```

Check your spelling

Typos can and will happen in the best of houses. Some browsers include a spellchecking engine, which can be set loose on all kinds of input fields. Enabling the spellchecker is as easy as setting the following extra attribute: `spellcheck="true"`

Stay compatible

Not every browser supports every part of each and every HTML5 specification. **CanIuse.com** is the semiofficial go-to point for developers wondering whether their feature of choice will impede compatibility with the common and not so common browsers too.

Don't get hacked

Keep in mind that any and all HTML5 components are pure client-side stuff. The availability of open source browsers has enabled hackers to create an 'evil browser' which does not comply to standards - skipping server-side validation is a sure-fire way to ask for pain.

5 Tools

FIVE ADDITIONAL FEATURES TO MAXIMISE YOUR WORKFLOW

Keep them offline

Caching has been around since the glory days of primitive web browsers. HTML5's application cache makes the cache engine addressable. Your code can inform the browser about elements which should be cached. The most common use case involves the creation of web apps, which keep running if deprived of their internet connection. In addition to that, the API can also be used to cache images.

Web storage bingo

In many cases, persistable data consists of but a few small tuples. In this case, the Web Storage API is the tool of choice. It provides a simple KV-Store, of which values can be persisted across sessions or across invocations of the web application. Beware, however, that most browsers tend to put tight constraints on the maximum size of the storage partition.

Database ahoy!

Sometimes, databases are needed to handle large and/or more complex data scenarios. The IndexedDB API provide an object oriented NoSQL database which can be used even in offline mode. As with web storage, be aware that the amount of space available to your application is likely to be limited in order to prevent local denial of service attacks by malicious or badly-designed websites.

Modify the back stack

Windows Phone 7 acquainted users with the **back button**. After having used one of Microsoft's handsets, users expect the back-button to work coherently. The History API permits you to manipulate the history of the browser. You can modify the address shown in the address bar, but you can also insert steps into the back queue just in case the user ever feels like backtracking through a process.

Even more technology

The web is evolving at an amazing speed. If you want to know more about new technologies, take a look at Mozilla's WebAPIs (developer.mozilla.org/en-US/docs/WebAPI) as it's very interesting to read. The company is in the unique position of having to maintain a completely web-based operating system. This unveils interesting problems normally not faced by browser vendors. Plus, Mozilla's drafts have been accepted as standard frequently.



Go 3D with WebGL

TAKE THE HARD WORK OUT OF CREATING 3D BY FIRING UP THREE.JS

Venturing into the third dimension has always fascinated programmers and designers. Even though 3D engines have been implemented successfully using nothing but JavaScript, their performance was incredibly low due to the lack of hardware acceleration. Fortunately, a technology called WebGL seeks to ameliorate this issue. On a supported browser, WebGL code is passed directly to the GPU and is run at speeds almost comparable to code written in DirectX and/or OpenGL.

Sadly, handling WebGL on its own is tremendously difficult: if you ever bothered yourself with creating OpenGL or DirectX rendering code, you are very well aware of the endless amount of pain hidden in this particular field of computer science.

Fortunately a nice wrapper framework called three.js is available. It acts in a fashion similar to XNA, providing

developers with a user-friendly frontend and taking care of the ugly and gory mathematical and computer scientific details transparently in the background.

While running WebGL on the desktop is by and large not an issue, support in the mobile space is spotty at best. For example Firefox OS was unable to accelerate WebGL scenes until the arrival of Firefox OS 1.2 – developers and users stuck on older versions of the OS need to live with agonisingly slow performance. In addition to that, the very popular Opera Mini browser also has its fair share of issues with WebGL. WebGL is tremendously cool, as can be seen by looking at the rendering code and the screenshot shown above. Sadly, its compatibility in the mobile space is limited – but since Adobe has abandoned Flash for mobile when it met resistance from the late Steve Jobs, developers have little choice.

Code library

For the full code in this library, check out our FileSilo

```
var renderer = new THREE.WebGLRenderer({ alpha: true });
renderer.setClearColor(0x0055ff,1);
renderer.setSize( window.innerWidth, window.innerHeight );
var loader = new THREE.JSONLoader();
loader.load( "Monkey.json", function( geometry )
{
    geometry.computeBoundingBox();
    mesh = new THREE.Mesh( geometry, new THREE.MeshNormalMaterial() );
    mesh.scale.set( 10, 10, 10 );
    mesh.position.y = 0;
    mesh.position.x = 0;
    scene.add( mesh );
    var pointLight = new THREE.PointLight(0xFFFFFF);
    ...
    for(i=0;i<9;i++)
    {
        var geometry = new THREE.TorusGeometry( 10, 3, 16, 100 );
        geometry.computeBoundingBox();
        var material = new THREE.MeshLambertMaterial( { color: 0xff00ff } );
        var torusMesh = new THREE.Mesh( geometry, material );
        torusMesh.position.x=2+getRandomNumber(30);
        torusMesh.position.y=2+getRandomNumber(30);
        torusMesh.position.z=2+getRandomNumber(30);
        torusMesh.scale.set(0.5,0.5,0.5);
        scene.add(torusMesh);
    }
    camera.position.x = 20;
    camera.position.y = 20;
    camera.position.z = 80;
    document.body.appendChild( renderer.domElement );
    function renderAll()
    {
        requestAnimationFrame( renderAll );
    }
}
```

1. Three.js data

Displaying data with three.js requires the presence of **renderer, a scene and a camera**. The scene is a keeper element for the various objects – 3D artists refer to them as models – which are to be displayed in the scene. Cameras are a digital representation of the human eye looking into the scene and onto its contents from specified position. Finally the renderer is responsible for generating the picture seen by the virtual camera camera using mathematical processes.

2. 3D models

Displaying 3D scenes requires **models and lighting**. Our code starts out by using the JSONLoader on a predefined and relatively complex model created with Blender. When the loading is complete, the object is scaled to make it appear bigger and is provided with a set of position coordinates. Finally it is added to the scene, where it will be displayed later. In the next steps, similar processing takes place for the light source and the torus objects, which are to act as surroundings for the monkey.

3. Materials

The generation of the individual torii is interesting for multiple reasons. First, the corresponding geometry class generates a new skeleton of the desired element. This doughnut-shaped object is but a wireframe, bare of any physical appearance. Making it displayable is accomplished by adding a material. You should imagine MeshLambertMaterial to be like a skin of silk paper glued onto the torus. Then, a custom randomisation function is invoked to generate random coordinates.

4. Rendering

In the final step, the **renderer is set up so that the scene is actually shown in the DOM tree**. Here, the rendering must be requested in a relatively complicated fashion and that's because most browsers don't actually permit code to start rendering whenever it feels like doing so. Instead, a frame must first be requested for the rendering process to happen at all. A rendering handler will then be invoked when the graphics pipeline is ready to accept further data.



Power up HTTP

FIX HTTP AND REAL-TIME DATA EXCHANGE WITH WEBSOCKETS

Establish a connection

Communicating via WebSockets requires the presence of a connection. Fortunately, establishing one can be accomplished by creating a new instance of the WebSocket object. Pass in the URL or the IP address of the socket, and you're set to go.

```
function testWebSocket()
{
  websocket = new WebSocket(<url>);
}
```

Set up event handlers

Socket connections are asynchronous and event-driven by definition. Thus, the next step to WebSocket goodness involves the assignment of event handlers to the various members of the newly created WebSocket instance.

```
function testWebSocket()
{
  websocket.onopen = function(evt) { onOpen(evt) };
  websocket.onclose = function(evt) { onClose(evt) };
  websocket.onmessage = function(evt) { onMessage(evt) };
  websocket.onerror = function(evt) { onError(evt) };
}
```

Send some data

Transmitting information to the server is really easy: you can simply invoke the send function with a string or JSON object of your choice. Beware though that some browsers don't actually permit the transmission of objects, and so in that case, you can simply serialise them all by hand.

```
websocket.send("Hello World");
```

Receive server commands

Reacting to data coming from the server is as simple as using the onmessage event handler. Our snippet below prints the data to the console - when we are transmitting objects, the next step would be a classic application for deserialisation.

```
websocket.onmessage = function (event) {
  console.log(event.data);
}
```

Roll your own

WebSockets are supported by most server side development tools. Both Node.JS and Qt can be made to provide a WebSocket server right out of the box. Furthermore, third-party client libraries are available for all the major operating systems.

RESOURCES

THREE ESSENTIALS TO KEEP YOU CODING LIKE AN EXPERT

Website **HTML5 Index** html5index.org

The HTML5 Index is to JavaScript what the MSDN is to Windows Phone: a never-ending selection of interesting tid-bits, resources and other interesting stuff. The Index has helpful links on each term which corresponds with other technical terms. It even separates the terms by library for easy reading. It furthermore contains a detailed list of parameters and official API specifications which simplify coding.

YouTube channel **Google Developers** youtube.com/user/GoogleDevelopers

Even though the big G's channel can become quite Android-centric at times, it nevertheless provides an attractive selection of amusing videos teaching new technologies and tricks of the trade. There are informative tutorial series like Developer How To or there are videos more aimed at the latest updates available like the 2015 I/O keynote speech. Be prepared for a deluge of amusing content - but watch that data cap if you're watching on mobile!

Twitter **MozAppsDev** twitter.com/mozappsdev

Mozilla's developer advocates are a never-ending source of all kinds of interesting stuff on the open web. Be it a tutorial, an interview, some musings on a RFC draft, a feature on a new standard or library: joining their 30,000 followers is a decision which you will surely not regret. They'll even advertise for job vacancies on there if you think you'll fit the bill.

Graceful degradation

When working on end-user-facing applications, an increase in compatibility tends to lead to a direct increase in earnings. Mobile developers have approached the problem of new APIs with gradual degradation. It is a straightforward technique based on the assumption that some features are not entirely necessary for application execution. Smart apps and websites offer their users a reduced featureset when run on legacy browsers.

Lock your mouse!

Pointer Lock solves the problem of being unable to control the cursor fully in web games. Using this snippet 'standardises' the vendor-specific property names with a common field: canvas.requestPointerLock = canvas.requestPointerLock || canvas.mozRequestPointerLock || canvas.webkitRequestPointerLock; This needs a call to requestPointerLock: canvas.onclick = function() { canvas.requestPointerLock(); }

Make it talk!

Really cool computers like Kit talk to their owners. The Speech API pulls off a similar trick, best accomplished like this: var sayWhat = new SpeechSynthesisUtterance("Let us say something!"); var voiceArr = window.speechSynthesis.getVoices(); sayWhat.voice = voiceArr[10]; sayWhat.pitch = 4; sayWhat.rate = 10; window.speechSynthesis.speak(sayWhat);

Listen up, Scotty

Some browsers also go the other way around using a proprietary server provided by Google. Their speech synthesis API usually takes a grammar, which is then analysed against the aural input. This mode cannot be used for free dictation - it is limited to formatted attributes from a known list. Due to the grammar in the background, it tends to be more reliable than free speech mode: entering village names, part IDs or airport codes should never be attempted in 'free scanning'.

GOOGLE ANALYTICS

Google Analytics is the most widely used web analytics service in the world and it's free. Find out the most important parts of the data to study and master key features to get more hits



“ Google Analytics is the window into your site’s data. Use it to understand who is visiting, what they are looking at, and what you can do to get them to stay longer ”

Know your audience

YOUR FIRST PORT OF CALL FOR GETTING INSIGHTS INTO YOUR AUDIENCE AND VISITOR BEHAVIOUR SHOULD BE THE ACQUISITION>ALL TRAFFIC CHANNELS REPORT

The Channels report enables you to see at a glance where your visitors are coming from according to Google's rules. Not all your traffic

will be captured by Google automatically, specific traffic such as Email will not automatically go under the Email channel.

Google uses their Direct channel as a catch-all mechanism for traffic that it just cannot categorise in any of its other channels. So it's important that you familiarise yourself with the UTM tags in order to make sure all campaigns you are running are accounted for.

To understand how Google uses these tags, it's important to get familiar with the concept of source - this could be Google, Bing, Facebook; and the concept of medium

- for example, paid search CPC, organic and so on. Google then uses rules based on these dimensions (not restricted to the ones mentioned) to define their channels.

For example the organic search channel is defined by all traffic that has a 'medium' exactly matching 'organic'.

The default channel grouping Google provides to you would cover most business' needs, but if you think you need a custom channel, Google enables you to create custom groupings based on these specific needs of yours. Just like the default ones, the custom channels are based on rules that you will have already defined in the interface utilising dimensions set out by Google Analytics.

Understanding key metrics

GET TO GRIPS WITH THE TERMINOLOGY USED IN THE GOOGLE ANALYTICS INTERFACE

Channels

Google's Default channel grouping splits traffic into eight groups and these are: Direct, Organic, Referral, Paid Search, Other Advertising, Email, Social and Display. You cannot change how these channels are defined but Google may evolve them in future. By using these groupings you will be able to clearly review visitor behaviour from a particular source and determine the channel's effectiveness without being muddled up by another channel.

UTM tags

If you're planning some activity that will drive traffic to your site - email or social for example, it's key that you identify this traffic and make sure you can understand and measure the yield of your efforts. Google uses custom campaign parameters called UTM tags to enable you to identify each piece of activity or 'campaign'. For example, these identifiable activities can include the source, the keyword or term used, the content and ads or the specific name of a brand.

By tagging the links to your posts, for example in an email newsletter, you can ensure that this traffic is correctly identified as it comes through. Google has a handy URL builder tool that makes it very easy to identify the most effective URLs.

Real-time

Once you have tagged your campaign URLs you can test everything is working as it should be by looking at the Real Time analytics report by going to Real Time>Traffic Sources. By clicking through to your site using your newly tagged link, you should see this visit showing the correct medium and source in the report if you've done this right.

You will then be able to monitor spikes and trends as they happen on your website.



Sessions

The number of periods in which any user interacts with your website. If a user leaves the site and returns after 30 minutes or more, a new session is recorded.

Users

The number of unique users who have visited your site for at least one session during the selected date range.

Bounce rate

The proportion of total sessions which consist of only a single pageview being generated before the user leaves your website.

Pageviews

The total number of page impressions during user sessions within the selected date range.

Explaining relationship

A session consists of a grouping of one or more pageviews or other interactions which take place on your site. A session ends once a user has been inactive for 30 minutes, or leaves your site and returns via a different channel.

In-page analytics

Accessed from the Behavior section, In-Page Analytics offers

insight into how users navigate around your site. A live website overlay indicates the proportion of clicks made from any page to each linked page, enabling you to establish the most popular paths taken to explore your content.

Audience behaviour

The Audience>Behavior report reveals how loyal your visitors are and how often they return to the site. It's possible to compare the behaviour and performance of new versus returning visitors and also see the number of sessions each user generated during the selected date range.

Best new features

GOOGLE ANALYTICS IS REGULARLY UPDATED WITH NEW FEATURES AND REPORTS, HERE ARE SOME RECENT ADDITIONS THAT YOU NEED TO KNOW ABOUT

Spreadsheets add-on

This handy Google Analytics add-on for Google Spreadsheets enables easy access to your data via the Analytics API. It's possible to build your own reports that query multiple Analytics views and manipulate the returned data.

With a little setup you can create your own regularly updated custom dashboards merging Analytics data with any other measurements and making use of the Google Spreadsheets charts to visualise the results.

Cohort analysis

Currently in beta and rolling out across Google Analytics accounts, Cohort Analysis splits your audience into distinct groups. These are based on a particular behaviour or attribute and enables deeper level analysis. There is no such thing as an 'average user' and at present much analysis does not distinguish between differing behaviours, instead bundling all users into a single pot. Each grouping, or cohort, shares common characteristics and new trends may be unearthed.

Benchmarking

The Audience Benchmarking reports enable your performance to be compared against aggregate data from other sites of a similar size in the industry and country. You'll be able to see how your traffic sources and engagement metrics compare to other sites in the sector and discover areas where you outperform your competitors and those where you underperform. To access benchmarking reports you need to agree to share your data anonymously with Google.

Audience groups

Google's DoubleClick advertising technology collects anonymous information about web surfers and shares this in Google Analytics. This enables you to discover more about who is actually using your website. It's possible to view an estimated age and gender breakdown of site visitors, user affinities and the products and services users are seeking through In-Market Segments. Your audience can be segmented by these filters to enable you to find your most valuable user types.

Content Experiments

Content Experiments enables easy webpage split-testing which can help when working towards improving the performance of your website. When setting up a content experiment you must define a goal or conversion metric for which you are looking to enhance. For example, an eCommerce site could monitor revenue or the total number of transactions, whilst a lead generation site could look to increase the number of overall completed form fills.

Intelligence Events

Intelligence Events are a commonly overlooked feature. Analytics continuously checks for metrics that have seen statistically significant variations over recent days or weeks and highlights these anomalies. Intelligence Events help to surface any unexpected changes which could be indicative of a problem or a success on a particular page. It's also possible to set up automated email or text alerts which are triggered when a particular metric changes outside a defined threshold.

Plan your content

When it comes to creating content for your audience, making a decision can take up time. Luckily, it doesn't have to be this way.

Best performing existing content: pageviews

Use the 'All pages report' to check your most popular posts. If one topic outstrips the others in pageviews then see if you can address another aspect of the same topic.

Most engaging existing content: time on page

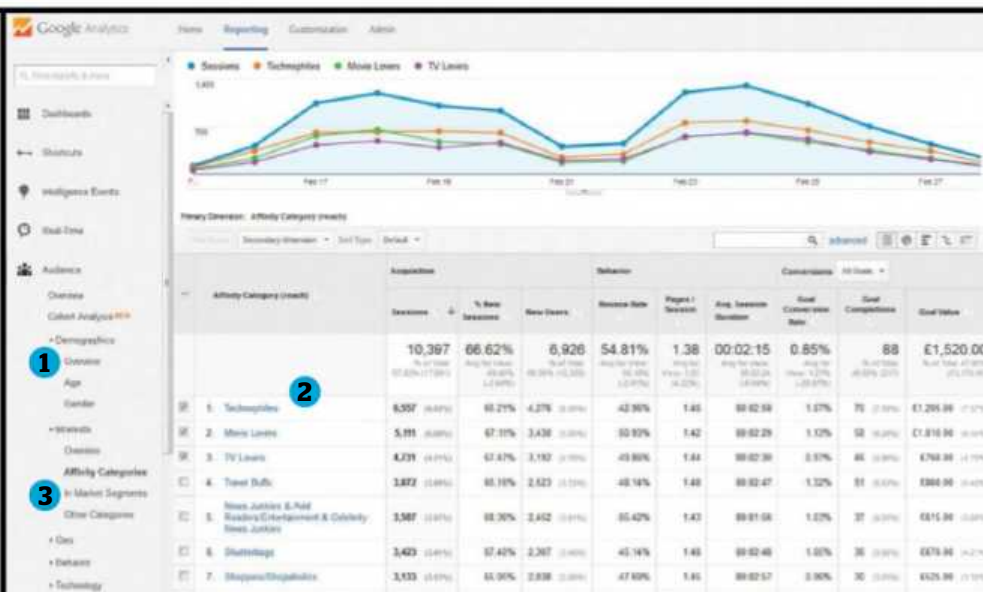
If you find an unusually high time spent on a post then write another with a similar angle, and interlink for the same effect. Use 'All pages report' again to find this out.

Internal site search

This one is straightforward. Found in the Behavior tab, the site search lets you know what your visitors have been looking for within your website.

Affinity Categories

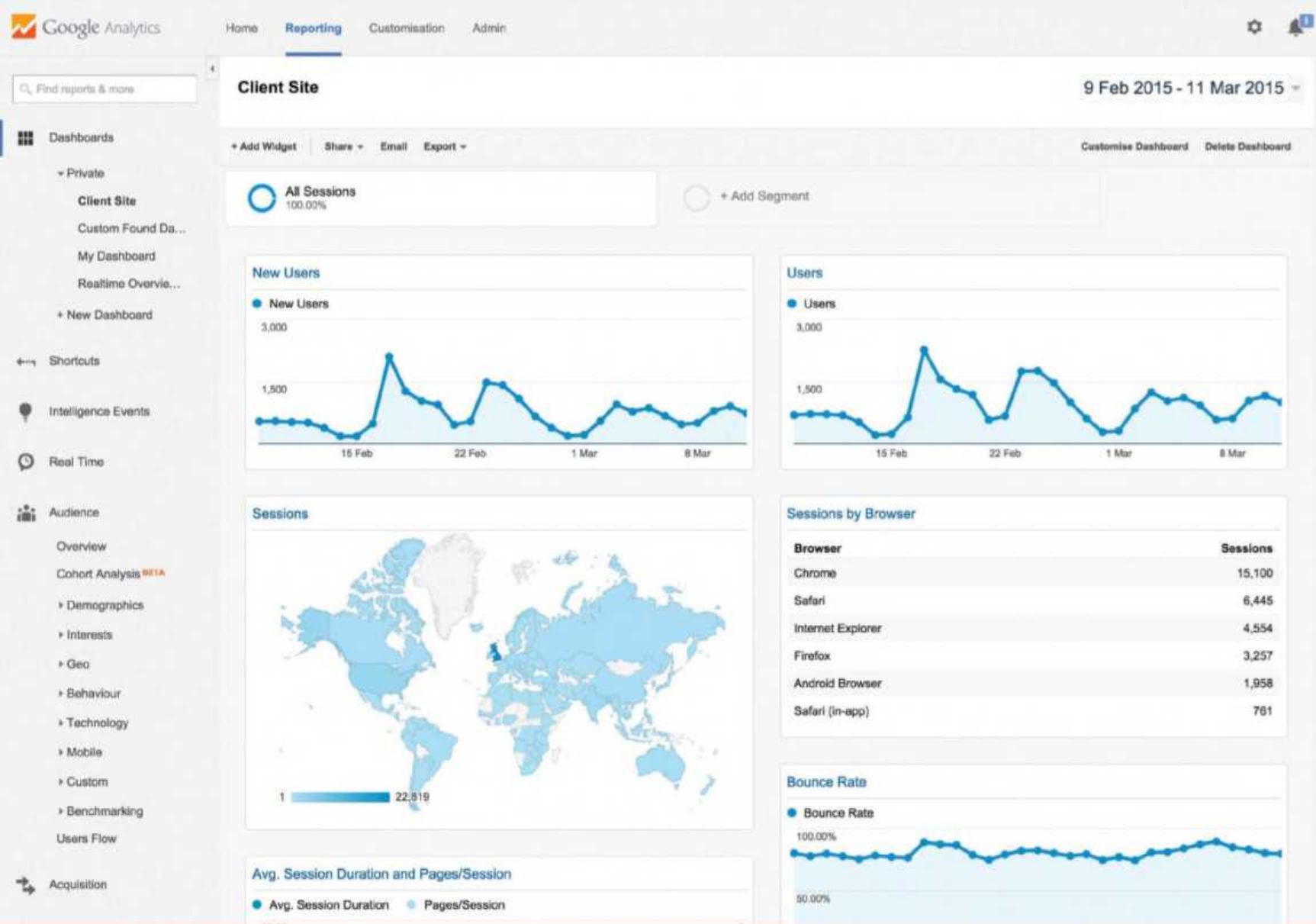
The Affinity Categories report can reveal your visitors' strongest interests. Including a post on a related interest and referencing effectively can provide new engagement.



1. Demographic reports can be viewed to reveal your visitors' data, like their gender split and age groupings.

2. User Affinity Categories can give insight into what else your users do online. Affinities reveal a more detailed interest breakdown.

3. The In-Market Segments report highlights areas of commercial intent in which your users have expressed an interest.



Customise the Google Analytics UI

Intelligence Events and custom alerts

GA generates automatic web alerts whenever there is a significant change in usage or traffic metrics. It can generate AdWords alerts in the GA interface if your accounts are linked. You can also set up custom alerts to be generated when traffic reaches or decreases to a certain threshold. Google enables you to receive these alerts via email or text message so that you can stay in the know even though you are not checking your GA reports.

There are four reports Google has available in terms of Intelligence Events: Overview, Daily Events, Weekly Events and Monthly events.

There are also some useful alerts you can set up like, if revenue drops by 10 per cent, if a landing page's bounce rate increases by 20 per cent, or if a PPC campaign performance increases revenue by 10 per cent, for example.

Custom dashboards

The usefulness of GA goes beyond the standard reports they provide - the interface is completely customisable and enables you to create bespoke dashboards and reports to match your needs. While most of the default dashboard provided out of the box may suffice, there are countless widgets you can create to really enhance your data analysis. Along with this, Google provides a great dashboard, report and segment repository called the Solution Gallery - a crowdsourced area where Google Analytics users publish their own dashboards, reports, anything that they have found useful for the benefit of fellow GA fans. This is a great shortcut and a brilliant resource too as the uploaded shortcuts are rated by stars by other users. You can download any of these premade dashboards to your own account and start using them straight away for your own site analysis.

Custom reports

Just like custom dashboards, GA enables you to create custom reports which are a very powerful tool in drilling down into your site data. Here you will get to choose and select only the dimensions and metrics that you want to display. This is so that you can easily and quickly access the data relevant to you without spending ages clicking around in Google Analytics each and every time you want that kind of specific data.

Google's help section is very useful here as it has a handy list of all the dimensions and metrics that GA uses, and also gives hints on the valid dimensions or metrics combinations as not all of them can be queried together.

As mentioned, the Solutions Gallery has a large repository of well-made, crowdsourced custom reports from other Google Analytics users that you can easily download, utilise and customise for your website's needs.

Automated emails

It's very easy to set up GA so that it can send you customised reports via email at a frequency that suits. This will then save you from having to go in and generate them each time and ensures that you are comparing like for like each week or month. These emails also act as a handy prompt to go in and have a further dig around in your Google Analytics data so that you're always on top of the latest trends on your website.

Intelligence Events reports can also be set up to alert you with an email when certain criteria are met. If set up correctly these can be utilised as an extremely useful warning system that will alert you to problems with your site or server that may need urgent attention, like if the bounce rate or page load time has increased significantly for example. Alternatively, the reports can be set up to make you aware of large traffic spikes that may warrant further investigation.

Internal site search

If your site utilises an internal search function to display a filtered selection of products or list of related articles, GA can track and report on this data.

The Site Search report, accessed under the Behavior section, provides details of the most commonly searched for keywords and the pages from which users make such queries.

A quick look through these terms will enable you to identify the content that users are actively seeking on your site. If this content exists, its prominence should be improved, if not you have a list that is all ready to go with suggestions for future content additions.

Search queries

Connecting your Google Webmaster Tools account to Google Analytics enables Webmaster Tools' off-site data to be combined with the on-site data in GA and unlocks the Search Engine Optimisation series of reports. The

most useful of these reports, Acquisition>Search Engine Optimisation>Queries, lists the different Google search queries for which your site has appeared in results. The data returned shows an estimate of the number of times your site was shown for each query as well as the average ranking position and click-through rate.

With most keyword referral data now reported as not provided, this report can give you visibility on the search terms that are sending traffic.

Pinpoint the best

As well as identifying pages that are performing below par, it's important to understand and learn from the success of your best performing content. If you can identify common themes in this content then similar content could be developed to bring in more traffic. The Behavior>Site Content>Landing Pages report reveals details of the sessions which result from a user landing via a certain channel.

Ecommerce and goal conversion data can be particularly insightful here as after landing on a particular page, users may go on to explore your site and convert at a different URL. It's possible that a page with a low average time on page may in fact be extremely valuable, for instance a category page which funnels users to products they then go on to purchase.

Determine the issues

IDENTIFY YOUR POOR-PERFORMING PAGES VIA PRESENTABLE DATA AND OPTIMISE

Bounce rate

Pages which have a high bounce rate are usually an indication that the content is not what the user expected to find. On eCommerce websites, product pages frequently have high bounce rates as users may land on them from a search engine when looking for a particular product. It is possible to decrease bounce rate on these kinds of pages by ensuring that all possible related products are interlinked.

A page that records a bounce rate significantly greater than other pages of the same type should be investigated further.

It's also possible to view bounce rate by channel, which may reveal users from social platforms are interested primarily in a single piece of content and less in exploring the rest of your site.

Time on page

As well as recording average session duration, GA also reports on the average duration of pageviews of each individual URL on your site. Pages can easily be sorted by the average time on page metric, and this can be accessed through Behavior>Site Content>All Pages to reveal the pages where users are leaving quickly. This data suggests that users do not find these pages engaging enough to stay on.

The Content Drilldown report aggregates page-level metrics by a site's URL directory structure, which can quickly highlight particular subfolders such as product categories that are under performing.

Low converting pages

If you've set up eCommerce or goal-tracking then Google Analytics will be able to report on a true monetary value of a page. Selecting the eCommerce option at the top of Behavior>Site Content>Landing Page will reveal a report on the number of sessions, transactions and revenue that has been generated through visitors who have landed on a particular page that you want to analyse. It's also possible for you to view the average order value, eCommerce conversion rate and per session monetary value for that particular landing page, which can flag up any poor-performing pages for your attention. This will then enable easier fixing later on.

Selecting a Goal Set option at the top of the same report enables conversion rates for different predefined goals to easily be compared.

Pages with low click-through rates

The Acquisition>Search Engine Optimization>Queries report can highlight pages which rank highly in search results but aren't delivering much traffic. It will identify a troublesome query, for which your site occupies a high average position but has a low click-through rate and this will suggest that your listing is not as enticing as other competing sites at encouraging searchers to go ahead and click through to your website.

You can effortlessly improve upon this by updating webpage title tags and meta descriptions to ensure they contain the related search query and a strong call to action. This is a quick-and-easy optimisation that should increase click-through rate and result in additional traffic.



“If you're not a regular Google Analytics user, spending just half an hour or so looking through the reports could reveal some easy opportunities to boost traffic”

Philip Gamble
Technical SEO manager, Found



“The Users Flow report is an overview of your best and worst performing pages. It's useful for gathering ideas on user journeys needing improvement”

Dora Moldovan
Head of technology, Found

PPC

If you are running a PPC campaign, it's important that you understand whether the traffic that is being generated is money well spent. By setting up goals in Analytics and assigning a monetary value to those goals, you can quickly establish the ROI on your PPC spend even if you aren't selling something, for example, a

subscription to your mailing list or a form submission can be a goal and assigned a value.

Remember that your PPC traffic should always have clear objectives and be directed at the most relevant landing page on your website. This will also need to be via the keywords that you are already using in order to give it the best chance at converting.

Vs

Social media

Social media traffic is altogether different from PPC. These visitors have generally arrived with far less intent to convert and have followed an interesting article or post that has led them to your site as oppose to an advert or offer. Understanding their behaviour however, is no less important. Do they

go on to read additional content on your site? Does a particular type of post generate more interactions than the others? Do they go on to find other content from your site and share that with their network of friends and followers? This information is invaluable for growing your audience and maximising the effectiveness of what you do for your site on social media and for building your brand.

Tracking your ROI - conversions

THE BEST WAY TO DETERMINE IF YOUR SITE IS SUCCESSFUL IS BY MEASURING YOUR GOALS

Once you set up the Google Analytics tracking code for your site, it's important to set up some conversion points for your site. Your business goals can vary - from tracking your online transactions to tracking whether someone has filled in your contact form, signed up to a newsletter or clicked to view a video.

Defining your conversions and understanding them in conjunction with your channel is key to your campaign success. For example, you may find that your paid search campaigns might be better at getting users to convert whereas your social media campaigns might be better at creating awareness of your brand and is a better traffic driver. It's important you define conversion points wisely and check your channels so you can understand your website or business' particularities and know where to channel your marketing efforts.

Types of conversions in GA

GA lets you set up different conversions for different needs. For example, Goals can be used for tracking key actions that occur on the site that usually reach a confirmation page, eg submitting a contact form, whereas Events is used to keep track of interactions that are actions typically tied in with page elements, like

clicking a button. GA provides a powerful API to enable merchants to track their website sales.

Goals

Google Analytics goals represent completed activities and conversions that measure your business objective success. Having properly defined goals enables GA to provide you with great insights on the effectiveness of your website page design or marketing campaigns. You can define up to 20 goals per reporting view.

GA lets you define your goals in several ways: Destination - where the goal is defined by reaching a specific page; Duration, a session that lasts a set amount of time or longer; Pages/Screens per session, where your goal might be to make sure the users see an x number of pages per session; and Event, where you can set up a goal based on an event, and this is useful if you want that event to be part of an attribution model like video play.

You can also define funnels for your goals and these are specific paths the users take to conversion. This is useful if you want to view where users abandon the funnel and where optimisation efforts should be spent.

Events

Events are user interactions that can be tracked independently from a page visit or load like video plays,

AJAX content, button clicks and so on. The difference between goals and events is that while goals are counted once per session, events are counted once per interaction. So if someone plays a video five times in a visit, that would register as five events. If they submit a form twice, that would count as a single goal. It is important that this difference is understood and the goals and conversions are implemented accordingly.

Another difference between goals and events is that the events need to be implemented programmatically via JavaScript code on the site, while goals can simply be defined in the GA interface. Using Google Tag manager makes goal implementation easy, so you should make sure that this is a consideration.

Ecommerce tracking

Based on information like products bought, transactions, and the time it takes for a user to purchase, you can get insights on your best-selling products or brands or categories, which channels are your best performers, your paid search ROI, and how long it takes customers to make a decision to purchase.

You can set up eCommerce tracking programmatically, but many kinds of shopping cart software will include GA eCommerce out of the box or provide easy-to-install extensions.

Glossary

Conversion

Any completed action that helps your business measure success. Conversions can be macro conversions or micro conversions depending on their importance. A completed purchase would be a macro conversion whereas something like a Facebook like would be a micro conversion.

Event

This is a type of user interaction with the page content. Events are utilised to track conversions when the interaction is independent on a page load. Examples of Events can include interactions like AJAX loaded content, link clicks, social buttons interactions, flash elements and so on.

Goal

A type of user interaction with the website. This is usually measured by the user reaching a specific URL or closely tied to a page load. Examples of this would be a user reaching some form of a 'Thank you' page, a purchase completed or a specified amount of time spent on a specific page.

Segment

A subset of sessions or users that share common attributes. Segments enable you to isolate and analyse groups of sessions or users for better analysis. You can apply up to four of these at a time and you can use predefined segments or import them from the Analytics Solution Gallery.

Channel grouping

A roll-up of traffic sources in the Acquisition reports that groups several marketing activities together. Channel groupings let you compare aggregated metrics by channel name, individual traffic source, medium or campaign name.

Attribution model

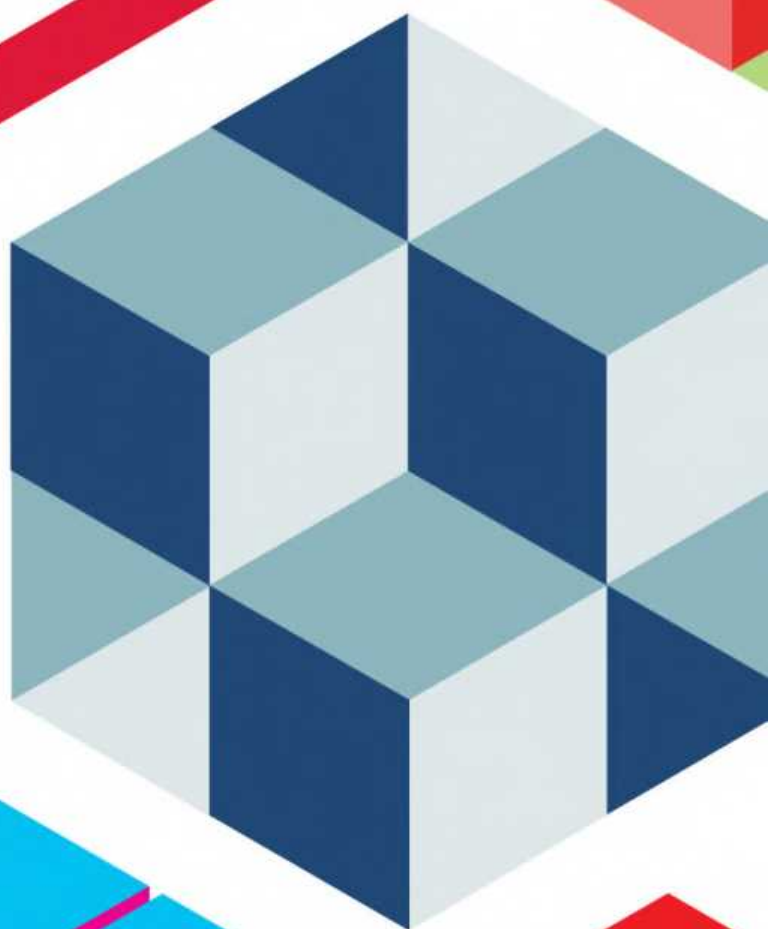
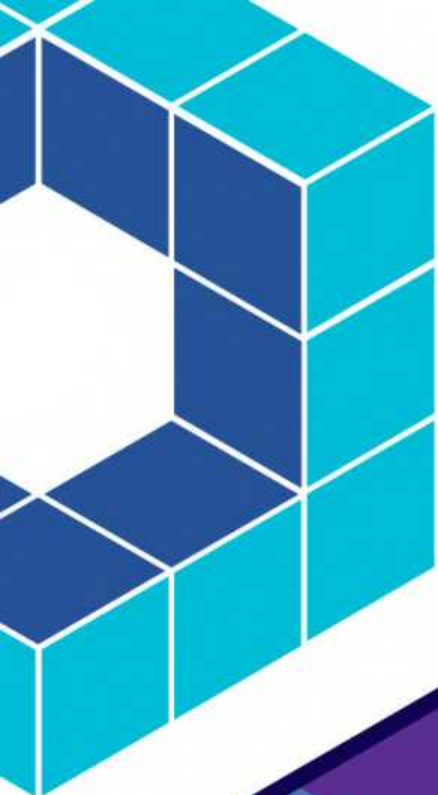
An attribution model determines which channels your sales and conversions get assigned to. This is based on contribution and there are many user models, for example a Last Interaction model will assign the credit to the last click before its conversion.

HTML & CSS

- 24** Power up CSS preprocessors
Master pro level tools
- 32** Set up your own grid system
using CSS
Build your own responsive grid
- 36** Make pop-up modal boxes with
pure CSS
Show new information in a box
- 40** Build a responsive fixed
page border
Always line your web pages
- 42** Design aspect ratio based layouts
Create ratio based layouts easily
- 46** Generate sweeping animations
with CSS
Make a swiper for presentations
- 50** Make a screen shrink on scroll
Reinforce your brand with ease
- 52** Produce an animated off-screen
3D menu
Make your menu stand out
- 56** Slide up titles on page load
using CSS
Put thought into content delivery

- 58** Animate scroll-reveal
split backgrounds
Create an inspiring layout
- 60** Animate an information card box
with CSS
Add text on hover
- 64** Construct an adaptive printable
CSS design
Make adaptable content







Why you need preprocessors

VETOING THE VANILLA CSS

Web Components are a totally new way of building website parts, instead of using HTML, CSS and JavaScript. CSS is not the simplest and smartest tool in a web designer's toolkit. Between vendor prefixes, baked-in constants for colours and fonts, random requirements for Retina displays, and the lack of a good grid system, CSS is half-finished at best.

But that doesn't mean you have to keep hand-rolling your stylesheets and suffer in silence when you're asked to change the colour scheme of a project at 5pm on a Friday just before a bank holiday. Preprocessors add some of the smarts that plain CSS never had. That means more creativity, fewer bugs, less typing, and a deep sense of joy throughout the entire design team.

The catch is there are three preprocessors instead of one, and they all have different strengths and weaknesses. You have to unlearn vanilla CSS and get used to writing preprocessor CSS – it looks similar but is different enough to trip you up. There's more thinking time when you're starting out, but that leads to less thinking later as you get used to the styles.

The not-quite-a-catch is that you can take the process further by combining preprocessors with an automated workflow that handles code formatting, CSS preprocessing, minification and obfuscation, and perhaps also final distribution – you don't need this stage but many preprocessor users do as it can save time.

But which preprocessor should you use? The right choice depends on the rest of your workflow. It also depends on which mixin features matter to you, and which tools and frameworks you use in the rest of your workflow. Choosing isn't hard, but you need all the facts. So that's exactly what you'll get over the next few pages...



“Preprocessors save you so much time when going from an idea to an implementation, by reducing the amount of code you have to write and providing some much-needed data-structures and logic to CSS.”

Max Glenister, front-end developer at Marmalade & Jam – @omgmog

What does a preprocessor do?

THE MAIN TRIUMPHS OVER USING REGULAR CSS FOR YOUR DESIGNS

A preprocessor takes a file of something that looks like a variant of CSS with extra features, and then converts it into a file that really is vanilla CSS. Isn't this pointless extra work? Not at all. It's a total win because you get to use a version of CSS with superpowers that you secretly always dreamed about but didn't know actually existed in the world.

Technically, a preprocessor makes it easier to write DRY – Don't Repeat Yourself – code. Examples? Here's a really useful superpower: you can define a colour scheme in one location and change it by editing a couple of lines. This avoids the usual insane search and replace that happens when your client decides the hot flamingo pink menu he wanted last week should really be canary yellow with orange text.

Doesn't that mean you can experiment more? It does. Suddenly big design changes get simpler, and you can play with styles instead of trying to wrestle them into submission in a sweaty and not very creative way.

How about vendor prefixes? If you're tired of typing `-moz-` and `-webkit-` in front of every animation command, you can leave the preprocessor to add them for you. Again, this means you get to think more about the look and feel of your site and less about where in the file you left the styling for the fifty-eighth div on the sign-up page.

As there are three popular preprocessors, you can choose what kind of superCSS you want to write. You can start small and keep changes simple, or you can go for the most different option from vanilla CSS, which still does the same things – only with extra superpowers.

Top Tip

For maximum productivity, consider using an equivalent HTML preprocessor. Haml, for example, can provide condensed, shorter and clearer HTML.

The big three



LESS lesscss.org

LESS was inspired by Sass, and arrived a couple of years later in 2009. It's considered mature and stable, but it still gets regular updates.

LESS includes variables, which enable you to set a value in one place and use it by reference everywhere; mixins, which are variables that drop all the content of a class into another with a single line; rule nesting, which clarifies inheritance; functions, which enable you to process numbers and strings; and plenty of colour management features. It includes basic search and replace and programmer-style text formatting.

LESS has good foreign language support, so it's worth considering if English CSS isn't your main interest. It also includes guarded mixins, which produce different CSS according to tests you can specify. It doesn't have fully general conditionals, which can be a limitation for some applications.

As it is used to define styles in the Bootstrap framework, you'll have to use LESS if you use Bootstrap. While developing, you can run LESS from the command line within Node.js or Mozilla's Rhino. You can also include it as a script in your pages and compile on the fly, but that's a bad idea for a production server unless your site gets a couple of hits a day.



Sass sass-lang.com

Sass (Stylishly Awesome Stylesheets) is a step up from LESS, with less (huh...) simple expansion and replacement, and more 'Let's pretend to be a developer' language features. Sass needs a language called Ruby, which needs some setting up on Macs, Linux, and Windows, although there's also a version that works on Node.js, with some differences.

Sass is more like three tools in one. You can write it as SCSS, which is an extension of CSS3 and looks like normal CSS with extra features, or you can write it as Sass, which uses indents instead of curly brackets. Both do the same things, but you can choose between familiarity and speed. The third tool is SassScript, which includes plenty of ready-made helper tools for manipulating numbers, strings, lists and so on. SassScript is quite clever. You can do things like define variables that work across a project or limit variables to the insides of a selector. It can even mix units, so you can calculate widths in a combination of ems and pixels, which is a neat trick.

Compared to LESS, Sass adds extra smarts, at the cost of extra learning time. It has more complicated and powerful conditionals and repeats, better selector management, and support for partials (or snippets of CSS you can pull in from a file).



Stylus learnboost.github.io/stylus

Stylus isn't as popular as LESS or Sass, but it may be underrated, and is certainly worth a look. One obvious big difference is cruft-free syntax, with no useless curly brackets or pointless semicolons, all replaced by white space and indentation.

Stylus is really the closest thing to a full programming language for CSS, this side of JavaScript. In addition to all the usual preprocessor features, like variables and operators, you can define your own functions for smart CSS parameter management and expansion. Stylus includes all manner of conditionals, loops, as well as an @ import option for reusing CSS libraries.

Transparent mixins are a big win. You can define functions in the usual way, but you don't need to specify a list of parameters. Stylus can copy any parameter string into the CSS. It even supports introspection, which means code can see where it is and what's happening around it, so it can behave differently in different blocks.

Another unique feature is interpolation, which is the ability to step through a predefined list of strings of values with a few lines of code, and to drop the strings/values into the final CSS. It's a good way to write dense, efficient code with the downside that it can be harder to read.

PROS

Great for newcomers

LESS looks a lot like CSS, but removes some of the more obvious annoyances. It's not a whole new thing that will make your brain melt, so you get to be productive right away! And because it's used with Bootstrap, which is used on so many sites, it's some way to becoming a standard.

Excellent support

Straightforward descriptions and plenty of examples make the LESS documentation easy to understand. Even with complicated features, you can work out what's happening. LESS is also better for debugging as messages tell you exactly where problems are so you can find and fix them.

CONS

Limited conditionals and inheritance

LESS can't do some useful stuff. Conditionals are limited, so there's no way to make really complicated conditional mixins. Selector inheritance is also limited. The & selector adds some basic inheritance, but it doesn't work with variables and it's not very flexible. There's also no way to mix units when you're working with maths.

No debugging options

Sass has debug directives, which print useful messages. LESS doesn't, so when something goes wrong you're stuck with guesswork, prayer, wishful thinking or – as a last resort – going through the code line by line until you find the problem.

PROS

Still CSS, kind of

Sass still looks like CSS, so it's easy to read. It's not quite CSS as you know it, and some of the more complicated features may make you stop and think. But it's good enough that it won't confuse you when you come back to a project a year after finishing it.

SassScript

The SassScript mini-language is like CSS on illegal steroids. Once you get your head around the fact that everything can be scripted, it's incredibly powerful and expressive. If you're more of a designer than a developer then this can be a shock to the system, but it's well worth the effort.

CONS

Needs Ruby

If you're a Ruby on Rails dev this won't be a problem for you. However, if you're more of a Node or Apache kind of person, then installing yet another development framework just to run Sass is going to feel a bit wasteful of time and resources.

Extra setup

As a result of its Ruby nature, Sass includes tons of flags and set-up options. You can use Sass out of the box without getting distracted by them, but they include a lot of useful features, and you're going to have to spend some time working out what they do.

PROS

Autocompilation middleware

Stylus includes Connect middleware. In plain English this means that Node can work out when you edit a file and rebuild your project for you. If you combine it with a task runner or some custom JavaScript code, you can persuade Stylus to build, compress and upload your project as you edit.

Lists and pattern matching

You're not limited to simple variables or constants. If you use a templating language, you can pass lists to Stylus and then you can have it generate CSS for each item. There are also true or false tests that are available for string searches and pattern matching as well.

CONS

Complexity

Stylus is very powerful, but it's not ideal for beginners. If you're comfortable hacking together a Node project, you'll feel right at home with it. If you're not, it's hard work with a steep learning curve and some lost productivity until you get up to speed.

Momentum

...Or lack of it. Although technically Stylus may be the most powerful preprocessor, it has lost ground to Sass, which is used by so many designers it's on its way to becoming a standard. This may not be a good thing, but there's no doubt Sass is more popular and has better community support.

Mixins, addons and extra help

MORE TOOLS FOR A BETTER WORKFLOW

csspre csspre.com

This is an essential overview site. It has a feature grid, a list of online and offline compilers and IDEs, and a further list of conversion tools. Short on details, but is a simple must-read.

Compass compass-style.org

A complete CSS authoring framework, Compass takes Sass and runs with it. It adds features from Ruby and better and smarter everything – including advanced typography, do-everything resets, various helper functions and CSS sprites.

Bourbon bourbon.io

Bourbon is a simple but powerful mixin library for Sass. It's lighter than Compass but easier to learn and just as useful for real projects. The docs are unusually comprehensive, and if you're using Sass you'll certainly want to look at it.

Lesshat lesshat.madebysource.com

Lesshat is a worthy mixin, with stand-out support for keyframes, gradients, fonts, transitions and more. The documents available on GitHub are pretty good too.

nib for Stylus nibstyl.us

A small library of ready-made CSS3 mixins for Stylus. Stylus has a lot of mixin goodness already, but adding nib makes it easier to write CSS3 one-liners for gradients, transparency, responsive images, border and size control and more.

Top Tip

If you have a standard workflow, use a popular IDE for compiling CSS. They'll do minification, gluing together, and other preflight jobs, without the hassle of custom code.

Client vs server, which is best?

THE OPTIMUM WAY TO COMPILE CODE AND SERVE THE FILES YOU NEED

Preprocessor code doesn't have to be precompiled. You can include a preprocessor as a script in your main site code. The usual trick is to put your preprocessor files into a special directory, add a stylesheet directive to tell the browser which preprocessor to use, and include the preprocessor itself as a JavaScript file.

Will this work? Yes. Should you do it? No. Or at least, not for production code. It doesn't take a lot of thought to understand that you're serving files you don't need, weighing your server down with extra work and slowing down page loads.

Compiling CSS can take a while. If you force the compilation, you're forcing users to wait for every page that uses a preprocessor script.

Worst case is this can add whole seconds to the load times. That's very bad, and certainly something you want to avoid.

There's also the CORS – Cross Origin Resource Sharing – problem. On a simple site, all the resources will be local. On a more complex site they may be served from different locations, and site loads can go horribly wrong.

As a rule, it's much smarter to precompile your code to vanilla CSS and minify it too to save download times. You get snappier page loads and happier visitors. The only extra cost is a compile or upload stage when you make changes. But if you have an automated workflow set up, that shouldn't cost you much extra time or effort.

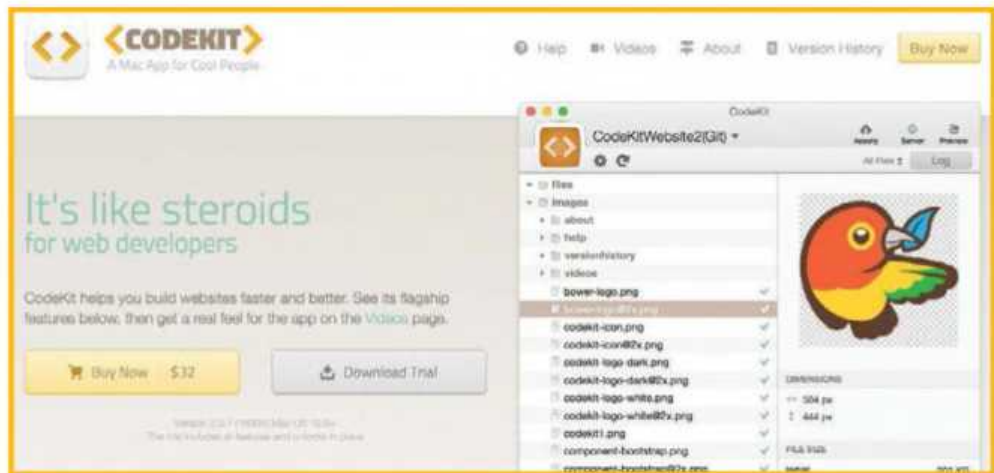
It's fine to use preprocessor scripts for testing and development. It saves you time while you're making changes because you can skip the compilation stage and leave it to the browser. But production code should be precompiled. Always.

Top Tip

Remember – your goal is to create superb, silky smooth and stylish websites. A preprocessor will save time, but you'll get a bigger benefit with the right framework.

Best compilers and IDEs

PREPROCESS WITHOUT PAIN WITH THESE COMMAND LINE-AVOIDING IDES AND COMPILERS



CodeKit incident57.com/codekit

For the not-so-epic price of \$32 – just over £21 – CodeKit gives you a tool that compiles everything from an editor window. LESS, Sass and Stylus are just the start. It also handles CoffeeScript, TypeScript, JavaScript, Slim, various HTML preprocessors including Haml and Jade, oh – and Markdown.

And there's more! The Foundation web framework, Bourbon, and Compass are built in, as is Autoprefixer. So you get to use all of that mixin goodness with almost no effort.

And the Sass compiler is the superfast libSass, which means you won't have to wait around for vanilla CSS to come out the other end.

If you want a good JavaScript editor, CodeKit automatically checks your JS for errors, so you won't

spend hours hunting down missing semicolons. If you have no idea what most of these do, you can ignore them. If you do, you'll love the fact that you get everything in one place, with no need to drop down to the command line or set up your own task runner. Minification and file joining are also available, so you can check distribution sizes before you ship. Lossless image optimisation is also built in for JPGs and PNGs.

It even includes a browser update feature, so changes automatically update in your development browser windows. The Bower component library is also included so you can download and install useful frameworks with a few mouse clicks.

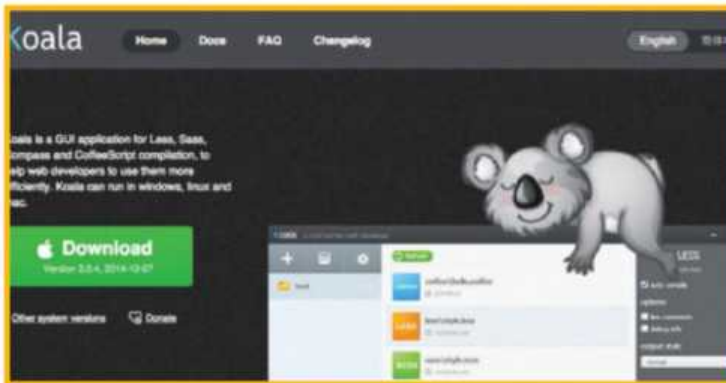
The downside? It's Mac only. Windows and Linux people are out of luck.



Run tasks with Gulp and Grunt

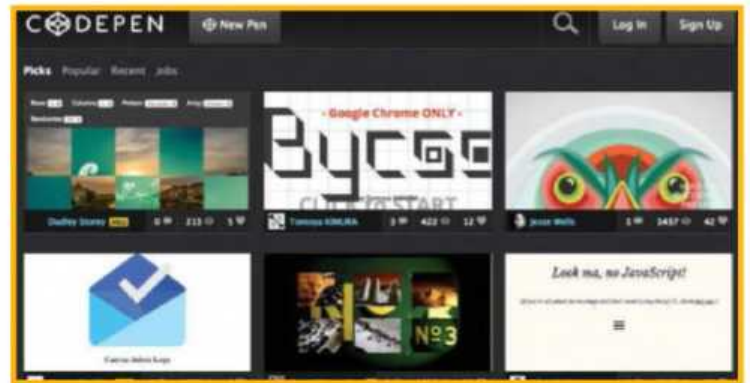
GETTING THE MOST OUT OF AUTOMATION

The hard way to use a preprocessor is to do everything from the command line, with hand-typed commands. The smarter way is to use some form of automation to precompile your preprocessed styles into vanilla CSS. Unfortunately automation is more complicated than it should be. If you're using Ruby on Rails, you can roll your own code. For Node.js users, the two tools of choice are Grunt (gruntjs.com) and



Koala koala-app.com

A freebie take on CodeKit, Koala works on Linux and Windows as well as on Mac. It can handle LESS and Sass (with Bourbon) but there's no Stylus support. So it's not as sophisticated as a commercial IDE, but it is cross-platform. And did we mention that it's also free?



CodePen codepen.io

For a different take on preprocessors, try Codepen's live code compiler. It's an easy to use but full-featured web designer's playpen. Choose between Sass, SCSS, LESS and Stylus, and select various mixins and addons using menus. Experiment with prefix-free syntax or add Autoprefixer support.



LiveReload livereload.com

Another compiler-of-compilers, LiveReload offers Sass, LESS, Compass, Stylus, and a slew of JS and HTML preprocessors - all-in-one handy package.

The price? It's \$9.99 (£7.99) in the app store for OS X. But the source is free on GitHub, so if you have a copy of Xcode and building a project from scratch holds no fears for you, you can make your own version - although if you do you should still pay for a licence anyway, just because.

Should you use it? It's maybe a touch busier and less straightforward than the competition, but if you're a code geek then you'll find it easier to customise (if that's what you need). It's certainly worth a look, anyway.



Mixture mixture.io

Mixture used to be a commercial product, but now it's been released for free.

It's a capable editor, template tool and preprocessor for Mac and Windows. It covers all your preprocessing needs with Sass, Compass, LESS and Stylus. You can even combine code written for different preprocessors in a single project by using a special Mix mode.

There's also support for Autoprefixing, script linting, debugging, minification and concatenation. This is not at all bad for a permafrees download, and certainly well worth a look so you can compare to it the paid-for competition and maybe save yourself a little cash.

Gulp (gulpjs.com). Both are task runners which can package up a collection of regular tasks you need to do into a mini-app you can run with a single command.

That means you can lint/hint, preprocess, glue together, minify, and maybe upload all your code with a single command. The catch is that getting to the point where the command works for you will take a while - a day or so if you know how to code, or a week or so if you're completely new to coding. But from then on, you can then use automation on all future projects - which means real timesaving.

Is Grunt better than Gulp, or vice versa? They're both useable, but they work a little differently. Some

people get on better with Gulp's syntax (instruction code), other people prefer Grunt's.

There isn't room here for a complete tutorial, but there are plenty of beginner resources online - like 24ways.org/2013/grunt-is-not-weird-and-hard for Grunt and travismaynard.com/writing/getting-started-with-gulp for Gulp.

And what about IDEs? There are a million and one - well, ten or so - online and offline editors and compilation tools that can take preprocessor code and turn it into plain CSS. They don't usually include full automation but they'll skip that time-wasting command-line stage, which is always a good thing.



Preprocessing in action

WANT TO SEE HOW PREPROCESSING WORKS IN THE REAL WORLD? TAKE A LOOK AT THESE EXAMPLES

Nesting code with Sass

SASS

```
a.myAnchor {
  color: blue;
  &:hover {
    text-decoration: underline;
  }
  &:visited {
    color: purple;
  }
}
```

CSS

```
a.myAnchor {
  color: blue;
}
a.myAnchor:hover {
  text-decoration: underline;
}
a.myAnchor:visited {
  color: purple;
}
```

With Sass you can nest items to group them together and make them easier to read. The magic `&` directive references the parent style

Here's the compiled CSS. See how Sass expands the nest and fills in the full references to the parent, but the code keeps all related elements in one place?

SASS

```
.first-component {
  $font-base: 1.4rem;
  .text { font-size: $font-base; }
  .button { font-size: $font-
    base+0.3rem; }
  @at-root .second-component {
    .text { font-size: $font-base
      0.2rem; }
    .button { font-size: $font-base;
  }
}
```

CSS

```
.first-component .text {
  font-size: 1.4rem;
}
.first-component .button {
  font-size: 1.7rem;
}
.second-component .text {
  font-size: 1.2rem;
}
.second-component .button {
  font-size: 1.4rem;
}
```

But what if you want to pull a style out of a nest so you can use it elsewhere? Here's some code to define some related text sizes

In the compiled code the `@at-root` directive flattens the nest so you can use `.second-component` elsewhere as it's not a subclass of `first-component`



CSS Tricks

@Real_CSS_Tricks

This is mostly autoposted content, but it's all design-relevant, and there are occasional gems of CSS wisdom to be found here.



CSS Preprocessors

@csspre

The clue is in the name - essential information and news for anyone who uses or is interested in CSS preprocessors.



Mark Otto

@mdo

CSS and preprocessors tweets from one of the creators of Bootstrap, with regular posts and news on future Bootstrap updates.



Harry Roberts

@csswizardy

Slightly technical industry tips and occasional tricks from this respected consultant CSS expert, author, conference speaker and top designer.



Sass

@SassCSS

Official Twitter feed of the Sass team - this is absolutely essential reading for regular Sass users who want to keep on top of Sass news.

Using inheritance with Sass

SASS

```
.message
  border: 1px solid #ccc
  padding: 20px
  color: #444
```

First, make a generic style. Note how with Sass you don't have to include curly brackets or semicolons - although you do need to get your indentations right

SASS

```
.ok-message
  @extend .message
  border-color: green
.error-message
  @extend .message
  border-color: red
```

Now you can use @extend to change some of the properties for special cases. These examples keep most of the original style, but override the colour

CSS

```
.message, .ok-message,
.error-message {
  border: 1px solid
#cccccc;
  padding: 20px;
  color: #444;
}
.ok-message {
  border-color: green;
}
.error-message {
  border-color: red;
}
```

Here's the compiled CSS. Sass puts back the curly brackets and semicolons, and it groups related items together, while still pulling out the override styles

Custom mixins with Stylus

STYLUS

```
border-radius(val)
  -webkit-border-radius:
val
  -moz-border-radius: val
  border-radius: val
button
  border-radius(5px);
```

The first part of the code defines a custom mixin and autoprefixer. Val just means 'plug something in here'. The second part uses the mixin to define a button

CSS

```
button {
  -webkit-border-radius:
5px;
  -moz-border-radius: 5px;
  border-radius: 5px;
}
```

Here's the compiled CSS. The "5px" is plugged into the mixin. You can reuse the mixin over and over, with whatever values or symbols you want to plug into it

PostCSS and the future of preprocessing

MAKE YOUR OWN WITH PLUGINS

PostCSS (github.com/postcss) is a processor that can do what you want it to. It's a combination of custom JavaScript and ready-made plug-in code for each feature you want to add is great in theory.

It's also part of the Autoprefixer autoprefixer tool (github.com/postcss/autoprefixer). Writing a custom preprocessor can be hard, but as plugins increase in number, PostCSS will get more useful.

Resources

THREE EXTRA TOOLS AND PREPROCESSING ALTERNATIVES

Chrome DevTools

developer.chrome.com/devtools/docs/css-preprocessors

It can be hard to debug preprocessor code, but DevTools makes it easy by including links to the original preprocessor source file and not the generated CSS file.



Myth

myth.io

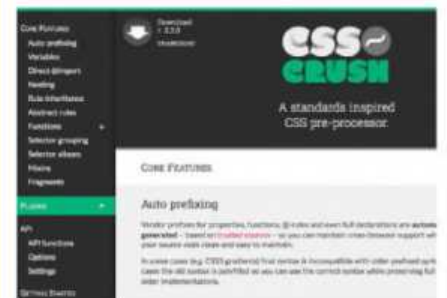
A simple but interesting alternative preprocessor with built-in future-proofing. Myth is simpler than the big three but worth a look anyway if you want simple, fast preprocessing.



CSS Crush

the-echoplex.net/csscrush

A standards-inspired CSS preprocessor written in PHP. You'll want this if you're a PHP shop, and Node or Ruby are of no interest. Also does minification, which is cool.



Set up your own grid system using CSS

Learn how to build your own simple, responsive grid for custom layouts

tools | tech | trends HTML, CSS



SS grids have been around for quite a long time – often they come bundled in frameworks such as Bootstrap, Foundation and Skeleton. These bundled frameworks are great, but

sometimes using a framework can be overkill if all you really need is a simple grid. The vast majority of

websites today use a grid. These websites may not intentionally have a grid system in place, but if they have a main content area floated to the left and a sidebar floated to the right, then we have a simple but effective grid.

If a more complex layout is needed, then people tend to reach for a ready-made framework just like the ones that we've already mentioned above. But CSS

grids are not difficult to create and anyone with a basic understanding of CSS could easily be able to create their own custom grid system. That is exactly what we're going to do in this tutorial. As you'll see later on, it's not hard or complicated and even making them responsive is no big deal. So open up your favourite text editor and let's create our own responsive CSS grid system.

1. Get started

Open up a new HTML file in your favourite text editor. Once all the meta information and link to an empty CSS file is added, we're going to add in a grid wrapper and call it 'grid-container' with an additional class called 'outline', which we will use to add some outlines to our columns.

```
<div class="grid-container outline">
</div><!-- END OF grid row -->
```

2. Use rows

Inside our grid container, we're going to add in another div with a class name of 'row'. This is going to be a generic class name and we will be using it a few times throughout our grid system. This particular row will be holding four columns and it wouldn't hurt to add a class name called 'column_four' for future reference, but we won't be using it for any styling purposes.

```
<div class="grid-container outline">
<div class="row column_six">
</div> <!-- end of row -->
</div><!-- END OF grid row
```

3. Add the four columns

Inside the first 'row column_four' div, add in four other classes called 'col-1'. These will be our four columns that will span across the top of our page. Inside these we will have text content, so put in <p> tags within the divs.

```
<div class="col-1"><p></p></div>
<div class="col-1"><p></p></div>
```

```
<div class="col-1"><p></p></div>
<div class="col-1"><p></p></div>
```

4. Add three columns

Now add in a new row with only three columns within. Again let's give our row an additional class name of 'column_three', which will help us so that when we do look at the HTML, we can understand things better. These columns will sit underneath our top six columns and we will give these columns a class name of 'col-2'.

```
<div class="row column_three">
<div class="col-2"><p></p></div>
<div class="col-2"><p></p></div>
<div class="col-2"><p></p></div>
</div> <!-- end of row -->
```

5. Add a two-column footer

The last section of our grid will include only two columns that will be positioned at the very bottom of our page as a two-column footer. Again we will give them a different class name of 'col-3' as these will be our third and final set of columns, and add in the <p> tags. That's it for the HTML, let's move on to the CSS.

```
<div class="row column_two">
<div class="col-3"><p></p></div>
<div class="col-3"><p></p></div>
</div> <!-- end of row
```

6. Type in content

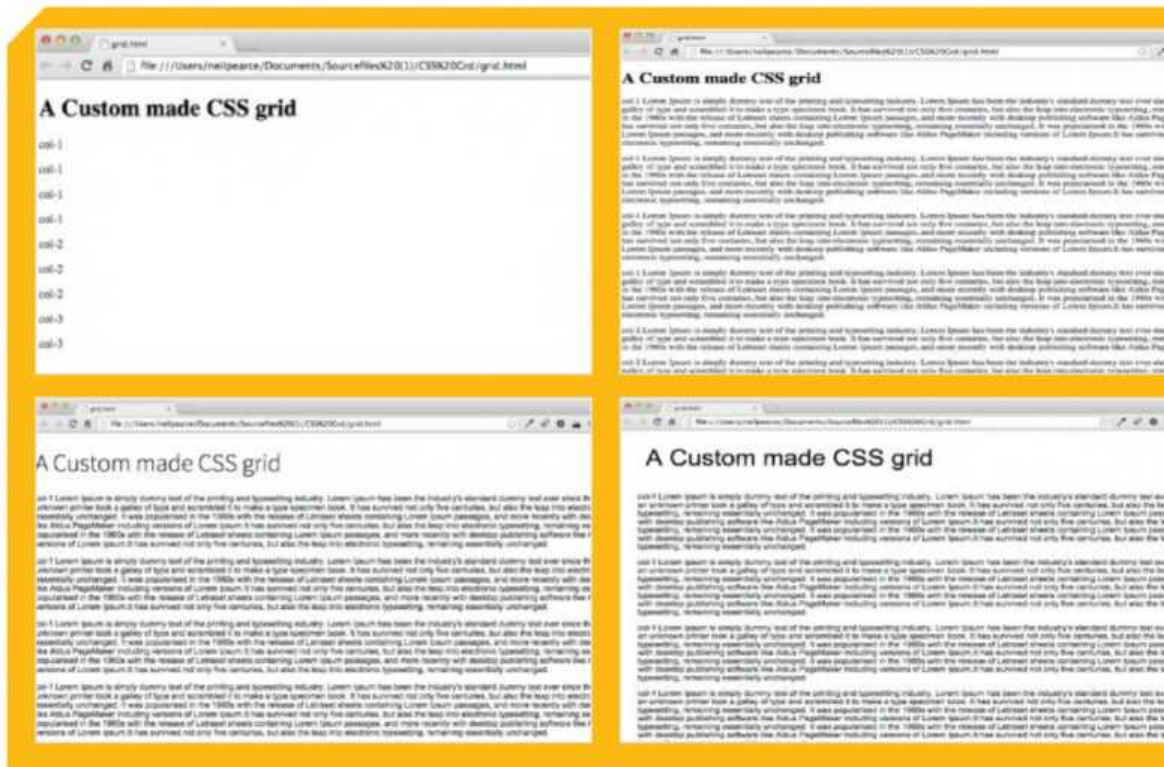
Before we start writing some CSS, we need to first put some content into our columns. So to keep things super

simple, copy and paste some Lorem Ipsum dummy text. However there's nothing stopping you adding your own content here. Just remember to also add some to the other columns (col-2, col-3).

```
<div class="col-1"><p>col-1 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s</p></div>
<div class="col-1"><p>col-1 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s</p></div>
<div class="col-1"><p>col-1 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s</p></div>
<div class="col-1"><p>col-1 Lorem Ipsum is simply dummy text of the printing and typesetting industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s</p></div>
```

7. Add the CSS

Go ahead and create a new CSS file and call it 'grid.css' and place it in its own folder called 'CSS'. As with most projects you start, the first piece of CSS you add will be a CSS reset. We're not going to go into the specifics here, but taking a quick look at the CSS rules should give you a good idea of what's going on.



- **<Top left>**
 - The grid is not looking all that clear at the moment, but we only have the bare bones of our grid
- **<Top right>**
 - With the content now added we can see things taking shape. Now it's time to move on to the CSS
- **<Bottom left>**
 - Now that we have the CSS reset added and some body styles, we can move on to actually styling the columns
- **<Bottom right>**
 - With the CSS now added to the grid container, and the clear fix rule added, we can see things taking shape

```
h1,h2,h3, body {
font-family: 'Source Sans Pro';
font-weight:300;
}
h1 { font-size: 3em; }
h2 { color:#2c2f36; font-size:2em; margin-bottom: 15px; }
html, body {
width: 100%;
padding: 0;
margin: 0;
height: 100%;
min-width: 100%;
max-width: 100%;
overflow: hidden;
}
a {
color: #888;
text-decoration: none;
}
```

8. Box layout model

One of the frustrating things about layout with CSS is the relationship of width and padding. After carefully defining your widths to match your grid, you then start getting issues when you start adding in text. So now you are subtracting pixels from your original width so the box doesn't expand, which is something we don't want to be doing. So adding the box-sizing property to all elements will give you the box model that you actually want.

```
*, *:after, *:before {
-moz-box-sizing: border-box;
```

```
-moz-box-sizing: border-box;
box-sizing: border-box;
}
```

9. Change body styles

In this simple step, we will add some styling to the font and the page background. We will keep things simple and choose an Arial font. The font colour will be black and we will make the page background white. Having it nice and simple like this will make it easier for us to see the gutters and outlines, which we'll take a closer look at in a later step.

```
body {
font-family: Arial, sans-serif;
color: #000;
background: #fff;
}
```

10. Get the width for the container

The purpose of the container is to set the width of the entire grid. The width of the container is generally 100%, but you might want to set a max-width for larger displays. A good maximum width to set is 1200px, but that's entirely up to you. And then we will centre the grid using the good old margin autotrick.

```
.grid-container{
width: 100%;
max-width: 1200px;
margin: 0 auto;
}
```

The outline property

The outline shorthand property sets all the outline properties in one declaration. The outline-color, outline-style and outline-width can all be set.

11. Utilise the clearfix hack

The main purpose of the row element is to keep the columns that are inside it from overflowing onto other rows. To make sure this is achieved on older browsers, we'll use the clearfix hack to make sure everything inside the row stays inside the row.

```
/*-- our clearfix hack --*/
.row:before,
.row:after {
content:"";
display: table ;
clear:both;
}
```

12. Position columns

There are many ways to position columns using CSS. But the most common way is to use floats. So by targeting all classes with a substring on col, we can target all our columns and float them left. To prevent them from stacking on top of each other, we will also give them a min-height of 1px and a 15px width gutter using padding.

```
[class*='col-'] {
float: left;
```


HTML & CSS



<Top left>

• Now we target all the p tags within our columns and give them default styles as well as 15 pixel padding to make a gutter

<Top right>

• With the column widths now added, we can see the grid taking shape with the columns positioned nicely

<Bottom left>

• We have now added in our outlines to show the grid in a nice visual way



<Bottom right>

• In this step, we will target all the paragraph tags again and align all the text and remove any margins or padding

[class*="col-"]

To apply CSS to a group of classes, you can use [class*="col-"] to select any element contains at least one occurrence of 'col-'

```
min-height: 1px;
/*-- our gutter -- */
padding: 15px;
}
```

13. Set the column widths

To find the width of our columns, divide the total number of columns by the width of our container. In our case, the width of the container is 100 per cent, and we want four columns to begin with, so 100 divided by four is 25 - our base column width (which will be col-1) is 25%.

```
/* Columns widths */
.col-1{ width: 25%;}
```

14. Column widths continued

Having now figured out how to calculate the first set of columns, we can now use that formula to work out the rest. We only want three columns in col-2 so we need to divide 100 by three, and you get 33.33. We will carry on until we get to 100%, and to make things easier for us in case we needed to add more columns later, we will add an additional two classes for col-4, and col-5.

```
.col-2{ width: 33.33%;}
.col-3{ width: 50%;}
```

```
.col-4{ width: 83.33%;}
.col-5{ width: 100%;}
```

15. Column outlines

To make things easier for us visually, we can add a 1px solid outline around all our columns, by targeting the outline class we added to the first row and using the outline property. This is really handy and will enable us to see both the main element and the padding.

```
.outline, .outline *{
outline: 1px solid #ddd;
}
```

16. Paragraph styles

To complete our grid, we will target all the 'p' tags within our columns and give them default styles. We will remove any margin and padding and then we will centre the text. Lastly we will set the font colour to black using the 'black' keyword. So that's our grid now completed.

```
[class*='col-'] > p {
padding: 0;
margin: 0;
text-align: center;
color: black;
}
```

17. Responsive columns

Adjusting our grid for mobile devices is pretty easy. All we need to do is adjust the widths of the columns. The only thing we need to be careful of is the last column in the row. This may want to hang off the end and to

counter this, we'll make the last .col-2 and .col-1 in the row 100% wide.

```
@media all and (max-width:800px){
.col-1{ width: 33.33%; }
.col-2{ width: 50%; }
.col-3{ width: 83.33%; }
.col-4{ width: 100%; }
.col-5{ width: 100%; }
.row .col-2:last-of-type{
width: 100%;
}
}
.row .col-5 ~ .col-1{
width: 100%;
}
}
```

18. Optimise for smaller screens

To finish off our responsive grid, we will target smaller devices. For this we will make all the columns 100% except for the first column. There won't be any positioning problems with our columns at this width, so there will be no need to add any additional CSS to counter anything.

```
@media all and (max-width:650px){
.col-1{ width: 50%; }
.col-2{ width: 100%; }
.col-3{ width: 100%; }
.col-4{ width: 100%; }
}
}
```

19. Add more columns

Just so we can see how easy it is to configure our grid,

we're going to add two additional columns to the top half of our grid, and fill them with some dummy text. So now that we have six columns spanning the page, we now need to make some adjustments to our CSS, which we will do in the next step.

```
<div class="col-1"><p>col-1 Lorem Ipsum
is simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been
the industry's standard dummy text ever
since the 1500s</p></div>
<div class="col-1"><p>col-1 Lorem Ipsum is
simply dummy text of the printing and
typesetting industry. Lorem Ipsum has been
the industry's standard dummy text ever
since the 1500s</p></div>
```

now have. Now that we have six columns, we need to divide 100 by 6, which now gives us 16.66. So within the col-1 selector, we need to change the width to 16.66%. So that's how easy it is to make adjustments to our grid.

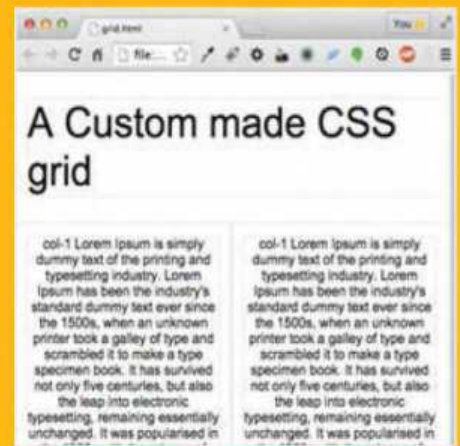
```
.col-1 { width: 16.66%; }
.col-2 {width: 33.33%; }
.col-3 {width: 50%; }
.col-4 {width: 66.664%;}
.col-5 {width: 83.33%;}
.col-6 {width: 100%;}
```

21. Final thoughts

Throughout this tutorial we have only created a basic grid. It is not a framework or a complete solution, but rather a starting point for creating your own CSS grid system. A grid system, however, is not for everyone, but creating your own will certainly help you decide if a grid is what you need.

20. Adjust the column widths

Just like before, we need to do some simple calculations based upon the amount of columns we



<Above>

• Now that we have added our media queries our responsive grid is looking as expected



The box-sizing property

Throughout this tutorial we used the box-sizing property, and to get a better understanding of this property, we will take a closer look at what it does. The box-sizing property is used to tell the browser what the sizing properties (width and height) should include. The CSS Box Model used to calculate widths and

heights of all elements isn't ideal. It's not broken, but it doesn't always play nicely within some browsers. So we use this property to emulate the behaviour of browsers that do not correctly support the CSS box model specification.

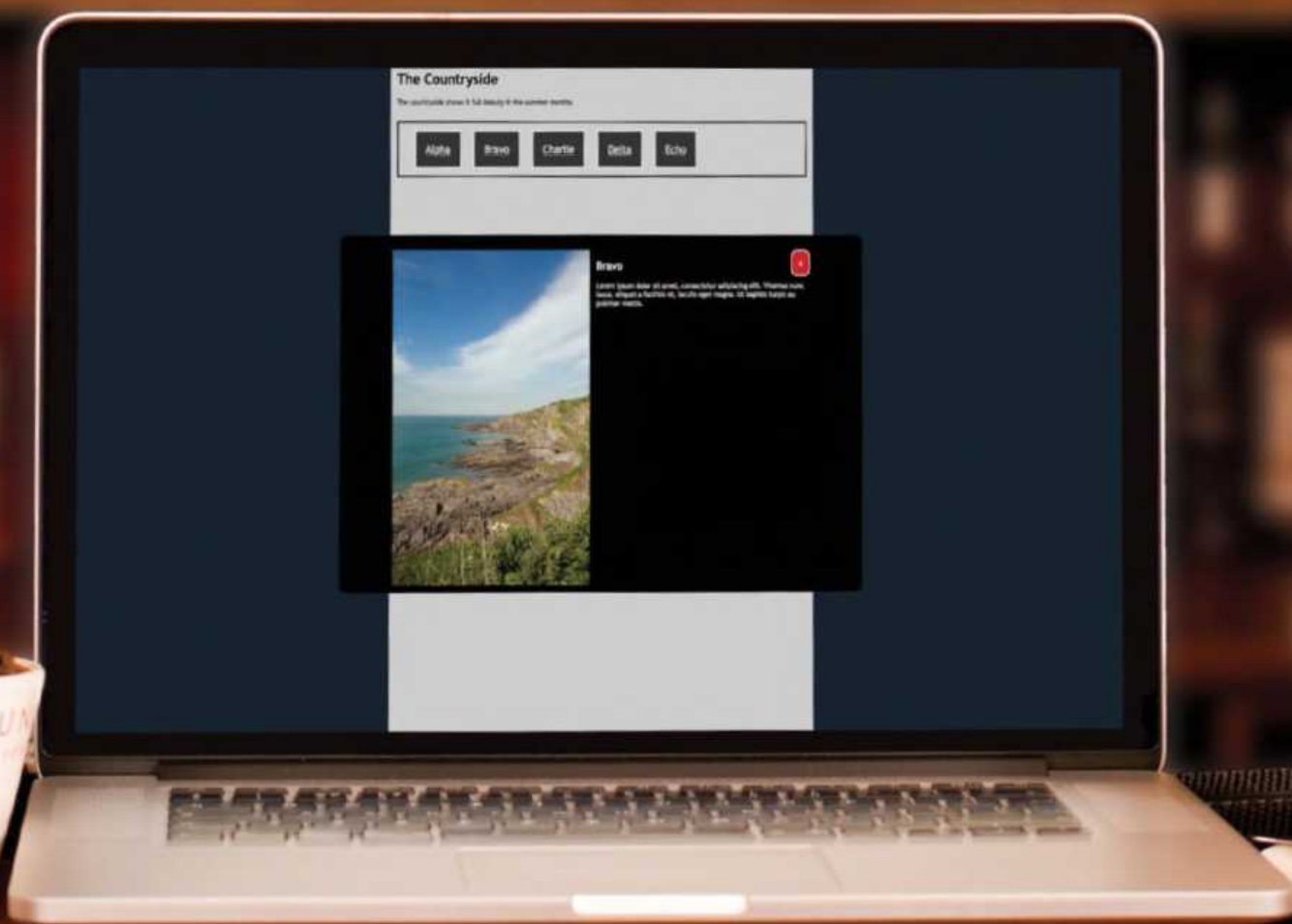
The default to this property is 'box-sizing: content-box;', but in this tutorial we used the value 'border-box'.

This makes the final rendered box the declared width and height, and any border and padding will cut inside the box and now we can safely declare our content boxes to be of 100 per cent width, including pixel-based padding and border and accomplish what we had originally set out to do without having to worry.

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Make pop-up modal boxes with pure CSS

Show new information in a modal box along with animation effects without loading a new page





ood design is always about presenting information in a way that is simple to understand. And yet, sometimes the amount of information causes a conflict with the need to present information in a

way that is easy to comprehend and where the information is still complete. One option would be to separate content onto different pages, but this causes scope for unnecessary page loading and poor usability, especially where users are accessing the webpage through a slow internet connection such as via mobile data with poor reception.

Modal boxes offer a perfect solution to the problem by enabling additional information to appear on the page when they are clicked without the need to load a new page. Furthermore, modal boxes can be closed again in order to return the user to the original content without the need to reload.

This tutorial looks at several options for how modal boxes can be developed with CSS to provide highly useable designs for websites and web apps that are easy to adapt for different types of content you may want to present. The functionality of these modal boxes will focus on easy access in a way that enhances the user experience of the webpage.

1. Get started

First, declare the main HTML page structure, including the head and body section within the HTML page. We will need a content container made from a div element with an ID called container to control page content flow.

2. Content overview

With the main layout elements in place, we can insert the main content inside the container element. The page will

have a descriptive title and introduction text to provide the main overview of the information contained on the page. This will provide information needed by the reader to identify whether they want to read further.

3. Option navigation

Next insert the available options that will provide the user with access to additional information. These options will be regular <a> links contained inside a <nav> container. Unlike regular <a> tags that use their href to link to external pages, these <a> links will link to ID components on this page using a # symbol in their href attribute preceding the ID name being linked to.

```
<nav>
<a href="#alpha">Alpha</a>
<a href="#bravo">Bravo</a>
<a href="#charlie">Charlie</a>
<a href="#delta">Delta</a>
<a href="#echo">Echo</a>
</nav>
```

4. Add modal boxes

The modal boxes will all use <article> as their container element and will have an optional attribute of data-transition to specify the type of transition they will use to appear and disappear, providing a high degree of flexibility for individual sections.

```
<article id="alpha">
</article>
<article id="bravo" data-transition="left">
</article>
<article id="charlie" data-
transition="right">
</article>
```

```
<article id="delta" data-transition="zoom">
</article>
<article id="echo" data-transition="fade">
</article>
```

5. Modal box content

Insert content into each of the <article> containers. Each modal box will need the ability to be closed, which will be triggered by an <a> tag linking to a blank ID using just # for its href. We also want to give this closing button a 'data-button='close' attribute so we can style it later.

```
<article id="bravo" data-transition="left">
<a href="#" data-button="close">x</a>

<h2>Bravo</h2>
<p>Lorem ipsum dolor sit amet.</p>
</article>
```

6. Initiate CSS styling

The additional CSS stylesheet resource file is still required to add the styling. Create a text file called 'styles.css', making sure that your text editor does not add .txt as a file extension. These also need attaching from the <head> section of HTML document.

7. Style HTML body

Don't forget that the <html> and <body> elements on a

Multiple transitions

Use data attributes to enable multiple transitions to be defined that can be easily added to individual modal boxes without having to overcomplicate your CSS.

The Countryside

The countryside shows it full beauty in the summer months.

[Alpha](#)

[Bravo](#)

[Charlie](#)

[Delta](#)

[Echo](#)

The Countryside

The countryside shows it full beauty in the summer months.

[Alpha](#) [Bravo](#) [Charlie](#) [Delta](#) [Echo](#)



<Above>

- HTML elements in place with content added to article sections, but no styling yet

<Far Left>

- The navigation container and its options are now styled to appear separate from the main content and identifiable as buttons

<Top right>

- Article sections acting as the modal box containers look like this without their styling

HTML & CSS

webpage are set to a height of just one line by default, hence causing problems if we want to create a container that has a full webpage height. Solve this by setting the `<html>` and `<body>` elements to be full height in the CSS.

```
html,body{
display: block;
width: 100%;
height: 100%;
margin: 0;
padding: 0;
}
```

8. Style the container

The container needs to be styled so that it will stand out. Change the main page background colour and position the container in the centre. To do this, we need to update the `<head>`, `<body>` and container sections.

```
html,body{
font-family: "Trebuchet MS", Helvetica,
sans-serif;
background: rgb(21, 34, 47);
}
#container{
display: block;
width: 1000px;
height: 100%;
background: #ccc;
margin: 0 auto 0 auto;
padding: 1em;
}
```

9. Options container

We want the options to stand out from any standard content on the page. You can do by styling the options

container that was created with the `<nav>` tag. This will be styled with a distinctive border and margin spacing to ensure it appears separate from surrounding content.

```
nav{
display: block;
padding: 1em;
border: 3px solid #000;
margin: 2em 0 2em 0;
}
```

10. Option styles

The option navigation elements need to appear like buttons, even though they are just `<a>` tags. This can be done through CSS by making them display as an inline-block, which enables a button appearance.

```
nav a {
display: inline-block;
padding: 1em;
margin-left: 1em;
font-size: 1.5em;
border: 3px solid #fff;
background: #333;
color: #fff;
}
```

11. Option hover

Good usability will make it easy for users to know which option they will select if they are to press on their trackpad or mouse button. We can apply the `:hover` property to the options so that they change colour when the user's mouse pointer is placed over them, indicating which option is about to be selected.

```
nav a:hover{
```

```
border-color: #333;
background: #fff;
color: #333;
}
```

12. Define modal boxes

The modal boxes all use `<article>` elements as their containers, hence we can use CSS to define their default styles for width, height and positing. We want these elements to appear at pixel-specific locations on the screen so we use absolute positioning to allow attributes for top, left and z-index.

```
article{
display: block;
position: fixed;
z-index: 9999;
left: 25%;
top: 100%;
width: 40%;
height: 50%;
padding: 2em 5% 0 5%;
border-radius: 1em;
background: #000;
color: #fff;
overflow: auto;
}
```

13. Set transitions

Modal boxes need to be able to show transition from one CSS property state to another for the animation to be visible. To keep our CSS simple, set 'all' properties to be eligible for transition animations. You can add other properties later to trigger their animation sequence.

```
article{
```



<Top left>

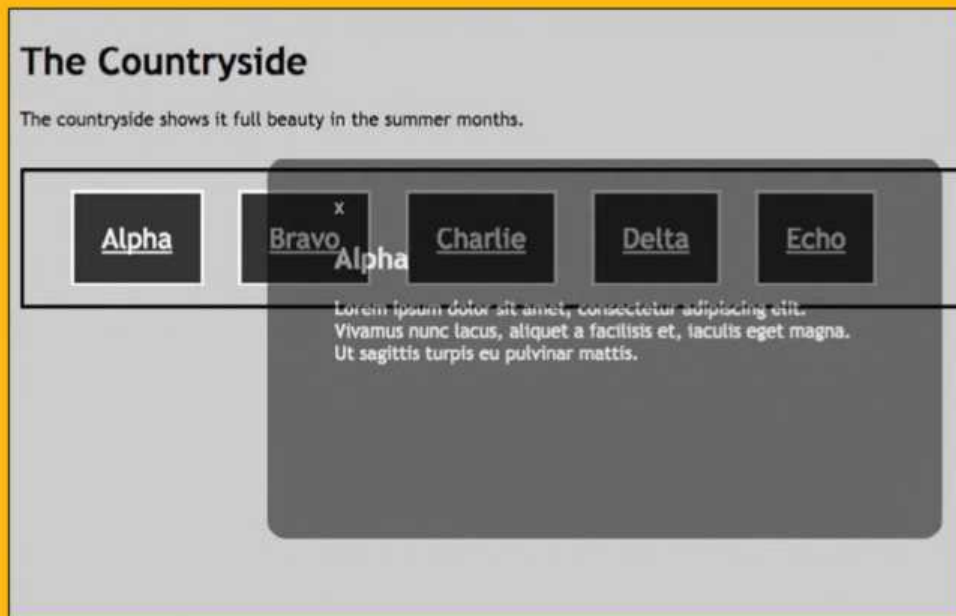
- Define the style of the modal box to make it clearly stand out from the main page content

<Top right>

- Modal boxes can also use transitions to slide in from the side when links to their associated ID are clicked

<Right>

- Alternatively, the modal boxes will now fade into view using transition animations



```
padding-bottom: 1em;
-webkit-transition: all 2s;
transition: all 2s;
}
```

14. Target selector

Now apply the 'target' selector to define the CSS states for elements that are selected via their ID. Elements will transition to this state when they are clicked on - ie this is the state that all modal boxes will appear as. The !important element ensures that these styles overwrite the previously defined styles for <article> elements.

```
:target {
visibility: visible !important;
left: 25% !important;
top: 25% !important;
width: 40% !important;
height: 50% !important;
opacity: 1 !important;
}
```

15. Transition: fade

Now that the modal boxes have the ability to transition from the default <article> settings to the :target settings, we can apply rules that will overwrite the default <article> settings through the data-transition attribute. Our first rule will let the modal box fade in and out of view.

```
[data-transition="fade"]{
left: 25%;
top: 25%;
opacity: 0;
visibility: hidden;
}
```

16. Transition: left

We want to let the modal box slide in from the left by placing the default position of the modal box as fully off the screen. This means that it will move from this position to the centred position when it is selected for display.

```
[data-transition="left"]{
left: -100%;
top: 25%;
}
```

17. Transition: right

We also want a transition effect to make the modal box slide in from the right by placing the default position of the modal box as fully off the screen to the right.

```
[data-transition="right"]{
left: 100%;
top: 25%;
}
```

18. Transition: zoom

The final transition to add is the ability for a modal box to zoom in and out from the screen centre. We'll also add

an opacity of zero so that the modal box will fade in and out of view as it sizes up or down.

```
[data-transition="zoom"]{
visibility: hidden;
left: 50%;
top: 50%;
width: 0;
height: 0;
opacity: 0;
}
```

19. Style close buttons

The modal boxes have an element with an attribute of 'data-button="close"' used to close the modal box. We will style this to appear in the top-right corner as a red circle:

```
article [data-button="close"]{
display: block;
float: right;
background: #c00;
color: #fff;
border: 3px solid #fff;
border-radius: 1em;
padding: 1em;
text-decoration: none;
}
```

20. Style images

Now we change the colour of the close button by applying styles through the :hover selector.

```
article [data-button="close"]:hover{
background: #fff;
color: #c00;
border: 3px solid #c00;
}
```

21. Image spacing

Our modal box design makes use of an image to be displayed at the left of the text content. We can use CSS to set the image as 100 per cent of the modal box article

```
img{
display: block;
float: left;
height: 100%;
}
```

22. Place a margin

Make sure that there is a margin of one text character to the right of the image so the text isn't too close. Place a margin to the right, so that the positioning of all elements in the modal box will take this into account.

```
article img{
margin-right: 1em;
}
article img{
display: block;
float: left;
height: 100%;
}
```



Modal box usage in UX design

Visual information often needs to take up a large amount of space in order to be fully useful. Check out Twitter, which uses modal boxes to display basic information about a person's profile that are triggered when you click on a Twitter handle. Using text or thumbnails to trigger the presentation of the full image or video when clicked provides a way to make full use of visual information without compromising the presentation of the main page content. Forms can also provide users with complementary functionality such as search, registration and account logins. Or, modal boxes can be used with alerts and notices - automated appearances of information can be provided to show important information that the user must see. Similarly, modal boxes for tips are presented as a clickable options for requesting additional information - useful for enabling users to access more information when they want it.

 **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Build a responsive fixed page border

inspiration studiodyxe.com

Having a simple and easily defined style is essential for any website and Studio D YXE, or Studio D as they are known has managed to create this effortlessly.

The key to achieving this has been to completely reduce the colour palette to a simple scheme of a light grey background, while a red focus colour appears throughout the site for all of the buttons and menu

elements. The red is then reintroduced for a stylised look to the photography.

The main text is a very dark grey, almost black colour and with this very reduced palette, all images fit together with the complete look that is stylish, elegant and easily recognisable. Here we show you how to achieve these kinds of results, with simple techniques that will make your design stand out.



<comment>
What our experts think of the site

Flexible design portfolio

"The client wanted the capability to create custom layouts for each case study, so we created custom building blocks in the backend where the client could simply check off which elements they wanted for a given piece. This resulted in each portfolio piece looking unique while keeping a beautiful aesthetic."

Dustin Weeres, creative director/owner, Island Collective

Simple navigation

On the homepage the navigation is hidden in an offscreen menu. As the user moves through content pages, the menu enables easier browsing and there are buttons for moving through the content.

Responsive border

The border around the edge of the page always fits the page exactly, no matter what size screen it's on and the rest of the content will then scroll below this.

Slide-in menu

There is a more traditional and familiar burger menu in the top-right corner of the screen that enables access to an off-screen menu. This menu then slides in from the top.

Back to the studio

The site logo found in the top-left corner also doubles up as another way to navigate. Clicking it on any page within the site will lead you back to the homepage.

Well-laid out menu

The menu is neatly tucked away to the left. As the user scrolls the menu stretches to indicate position on page.

Active content

Scrolling down the page causes other content to animate into the viewport from the left and the right as the user reaches those sections of the site.

Stand-out shades

The colour scheme of the site is kept to a light grey background, the text to a dark grey and a red accent colour is given to the main image to draw attention to the buttons and text.

Inspiration

Keeping content tidy

As with most sites that have scrolling content, images are animated onto the page as the user scrolls to that section of the site and they will then come into focus within the browser window. This keeps maximum impact for the animated elements as they are

presented to the user. The site gives the impression of loading content into the same page by animating a div over the top, then calling the next page and removing the div when that page has loaded, removing that annoying flicker that can occur when going from page to page. This effect has been achieved with the animation jQuery plugin. Read on to find out more.

Technique

The responsive border

1. Body content

The Studio D site has a great border that fits around the edge of the site and this is always there, no matter what content is on the screen. To create this, firstly an empty div tag is required with the class of 'wrapper', this will contain the border.

```
<div class="wrapper"></div>
```

2. For testing only

In order to test that this works as it should, we need some page content that's longer than the actual page. Here another div is added and this will be given a long height so that later on, the border can be seen working around the edge of the page.

```
<div class="long">A long piece of content</div>
```

3. Add the CSS

In the head section of the page, add an opening style tag so the CSS can be applied to the elements. The first styling will be for the page itself. Setting the height to 100%, the background to a similar colour to Studio D and a large padding around the edge helps get the look.

```
<style>
html, body{
height: 100%;
```

```
heroText">
background-color: #f0f0f0;
padding: 20px;
}
```

4. Style the wrapper

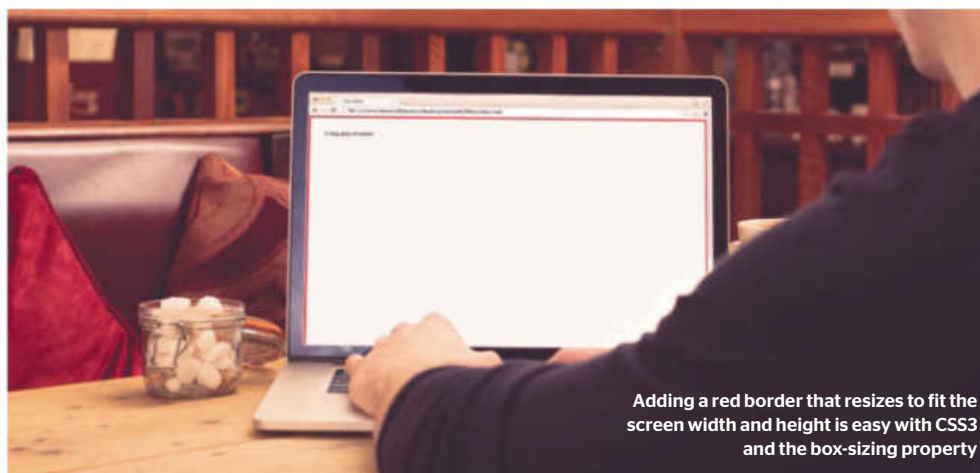
The wrapper class is going to contain the border, so it is set to be fixed on the page in the top-left corner. It's given a z-index higher than other page content and then made to fit the width and height of the page. The box sizing is set to include the border, then a 5px border is added all round.

```
.wrapper{
position: fixed;
z-index: 999;
top: 0; left: 0;
width: 100%;
height: 100%;
box-sizing: border-box;
border: #ff0000 solid 5px;
}
```

5. Finishing off

The class of 'long' is only here for testing purposes and it's given a height of 2,000 pixels which is larger than most monitors. Save this and test in the browser to see the red border always on the screen and fitting neatly into the browser window, even working responsively.

```
.long{ height: 2000px;}
</style>
```



Adding a red border that resizes to fit the screen width and height is easy with CSS3 and the box-sizing property

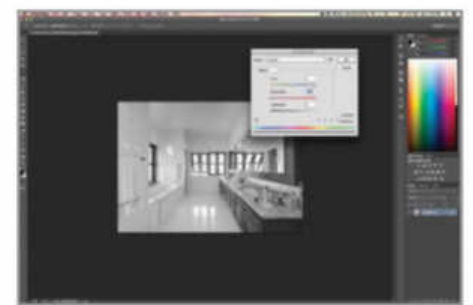
Technique

Create the red images

On the website the images have been treated so that they are all similar. This is good for the web because there is less colour information so images will load faster.

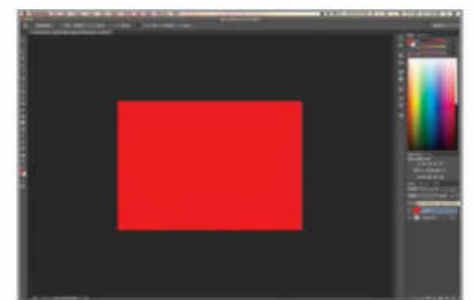
1. Convert to greyscale

Rather than converting it to greyscale and then back to RGB, just press Cmd/Ctrl+U to open the Hue/Saturation in Photoshop once you have selected your image. Drag the saturation back to lose the colour.



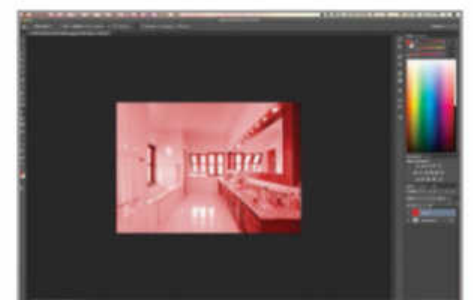
2. Add the colour

Create a new layer and make sure this is above the first layer. Choose a red colour or a colour of your choice and fill the new layer with this colour. By affecting the blending mode we will easily be able to get the look that we want.



3. Finish it off

Change the blending mode in the layer panel to Color and reduce opacity to 65 per cent. This lets some of the darker colours through so that you get a good tone. Now save this for web as a JPEG.



↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

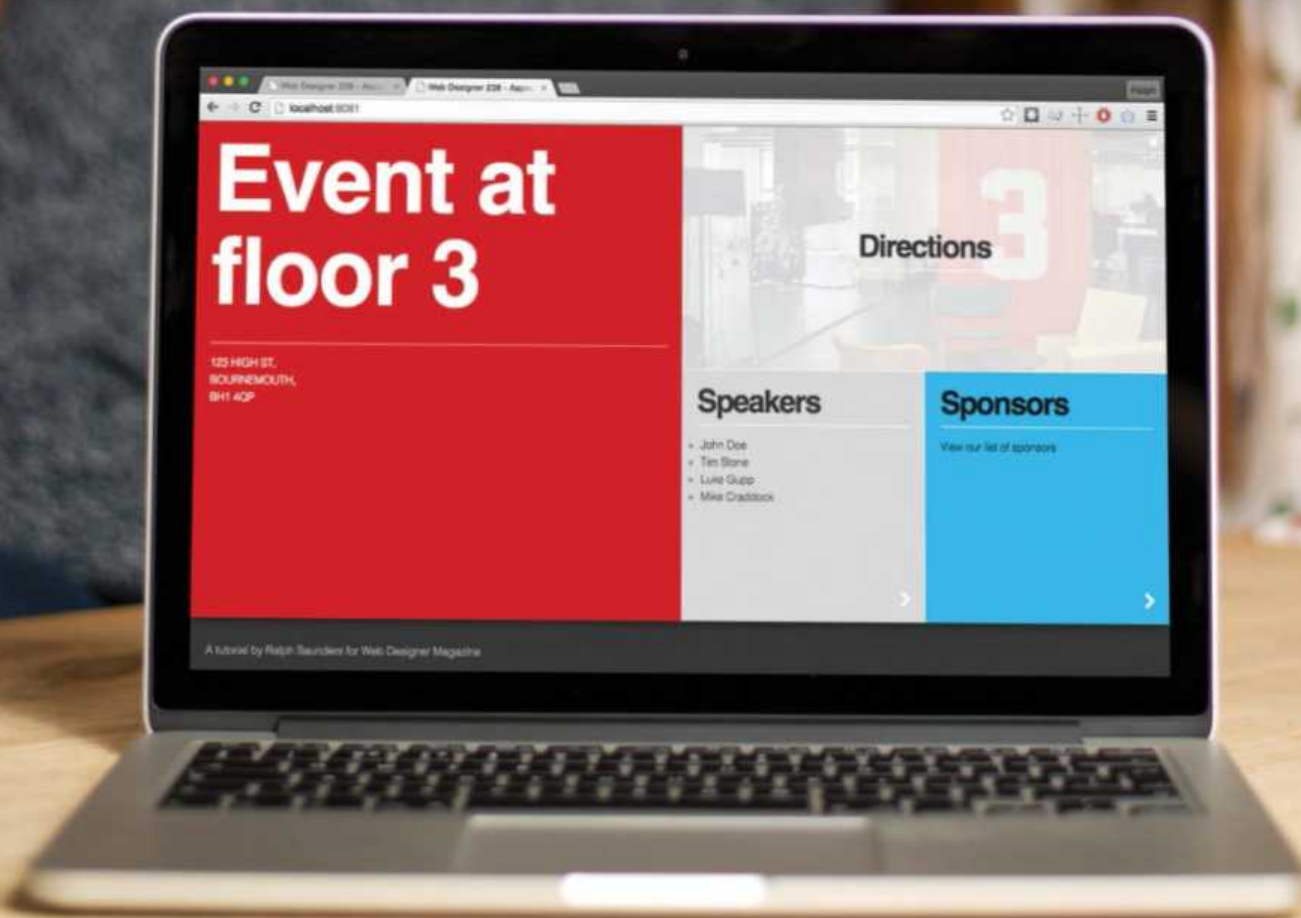
Design aspect ratio based layouts with HTML and CSS

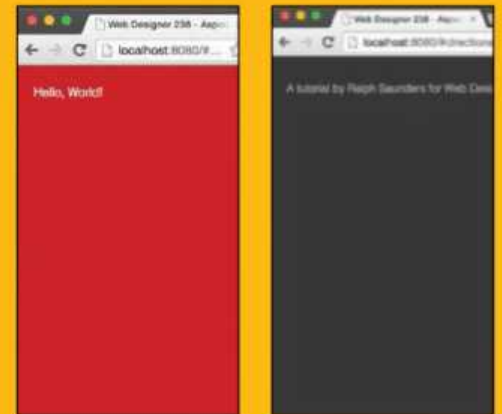
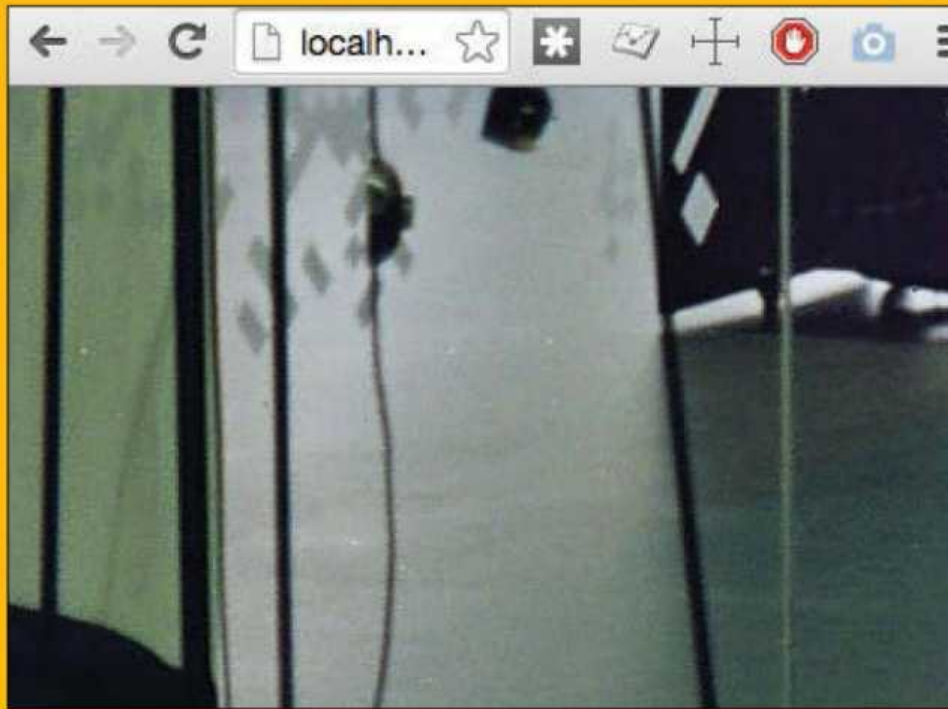
Building tile-based UIs can be complex. Discover how to make this process easy and create responsive aspect ratio layouts

Traditional web builds assume content is flexible and likely content managed. Boxes change in height as the page resizes because they're defined by a percentage width and their content. But this flexibility isn't always a requirement. We can craft more interesting layouts when content is well defined.

Aspect ratio-based layouts prioritise proportion over content. In this sense the process is much more like editorial – with all of its pros and cons. The aspect ratio approach gives us more design control but it's much more labour intensive to produce a good experience. If your client does not want to engage in editorial processes or requires flexible systems then steer clear! This approach is fantastic for scenarios where data is well

defined or can be truncated without issue or where iconography is more important than text. Dashboard pages, widget collections and even some web applications are all digital experiences that could benefit from this approach. Here we're going to cover how to build aspect ratio-based layouts, approaches for dealing with overflowing content, and overcoming the issues with making aspect ratios work responsively.





<Left>

- The image we've put in is far too large. Let's make it fit our page better

<Top left>

- This is the aspect-ratio technique. The rest of the tutorial focuses on working with other ratios and content

<Top right>

- This is what you should see when serving the files from the start-here folder

1. Download files

The ZIP file provided on FileSilo for this tutorial contains all the assets that we will be using along with a basic boilerplate setup. We're going to use an npm package called http-server to serve our files.

2. Mark up aspect ratios

The aspect ratio boxes don't have any semantic value in this tutorial, so we've used <div> tags. They could be <section> tags if appropriate. We're also using data attributes to semantically describe our 'aspect' element.

3. Style aspect ratios

We're using the ratio data attribute we created as a selector (supported by IE9 and up). Set the height to 0 and control the size of the box with padding only. As we're creating a square, the padding is equal to the width. We also hide any child content if it were to overflow.

```
.aspect[data-ratio="1:1"] {
  max-width: 100%;
  height: 0;
  padding-bottom: 100%;
  overflow: hidden;
  background: rgb(211, 22, 22);
  color: #fff;
}
```

4. Harder ratios

Using 4:3 is a little more complicated because we have to do some maths to figure out this ratio. The way to calculate how much padding you need to give an element is by dividing the height by the width and

multiplying by 100. So for example we would use in this case: $3 / 4 = 0.75 * 100 = 75$.

```
.aspect[data-ratio="4:3"] {
  max-width: 100%;
  height: 0;
  padding-bottom: 75%;
  background: rgb(223, 28, 66);
  overflow: hidden;
  color: #fff;
}
```

5. Style aspect content

The content inside any given aspect box will look bad if it sits flush against the edges of the box. We can get around this by giving a margin to the .aspect-content div, which should wrap all written content in a given box.

```
.aspect-content {
  margin: 20px;
}
```

6. Add imagery

Working with images in aspect ratios is tricky. We want the image to scale, but we don't want to distort it. With this in mind, we will need to write markup that enables us to crop it if necessary.

```
<div class="aspect" data-ratio="2:1">
  
  <div class="aspect-content">
    <h4 class="headline">Directions</h4>
  </div>
```

Compress images

The images provided in this tutorial are uncompressed - they weigh 4MBs each. Images can be compressed in Photoshop with the 'save for web' dialog or an npm module like imagemin.

```
</div>
```

7. Style the image

Here we position the image absolutely, relative to its parent container. We also tell it to fit its parent container with the max-width property and have it crop from the bottom right. We also want this image to sit behind everything, so give it a z-index of 0.

```
.aspect {
  position: relative;
}
.aspect img {
  max-width: 100%;
  position: absolute;
  right: 0;
  bottom: 0;
  z-index: 0;
}
```

8. Content to the front

We need to bring the content to the front so we're giving it a higher z-index than the absolutely positioned image and we give it the lowest possible z-index for this effect.



<Top left>

- Positioning the content over the image required both z-index and position-relative properties

<Top right>

- Flexbox works across evergreen browsers. For older browsers the text will just be aligned top left

<Right>

- Instead of modifying the image we simply use transparent backgrounds and animate the content that sits on top of it



Avoid styling

Large images with CSS styles on (like opacity) can take longer to render which makes animations slow. Try avoid applying styles to images and instead animate simpler elements.

```
.aspect-content {
  z-index: 1;
  position: relative;
}
```

9. Link the boxes to other pages

Here we take the 2:1 box and wrap it in an anchor element. In HTML5 this is valid so long as there isn't an anchor within an anchor. We also give this anchor a class, which we will use to change styles specifically to this box.

10. Style directions

We're overriding the normal .aspect-content styles so the content area fills the box and resets margins. Flexbox brings the content to a vertical and horizontal centre.

```
.directions .aspect-content {
  width: 100%;
  height: 100%;
  margin: 0;
  position: absolute;
  display: flex;
  align-items: center;
  justify-content: center;
}
```

11. Animate hover on directions

We want the hover state to change the entire tile as the area is clicked. We've created a cubic-bezier transition for a nicer animation and all attributes that change between normal and :hover states will animate using this.

```
.directions .aspect-content {
  color: rgb(55, 55, 55);
  background-color: rgba(255, 255, 255, 0.83);
  transition: all .5s cubic-bezier(0, 1.14, 1, 1);
}
.directions:hover .aspect-content {
  background-color: rgba(47, 47, 47, 0.8);
  color: #fff;
}
```

12. Animate the image hover

Fading the image out on hover will pull focus to the text and therefore the action of the box. We use the same animation and timing as the other hover effects on this element. The short duration is important because we're used to state changing instantly on hover. Animations over .5s feel sluggish.

```
.directions img {
  transition: opacity .5s cubic-bezier(0, 1.14, 1, 1);
}
.directions:hover img {
  opacity:.5;
}
```

13. Bring it together

Now we've got the different aspect ratios and some styles together we need to pull it together. Because the aspect boxes are fully responsive we can simply wrap them inside a div that limits their width.

```
<div class="one-half">
  <a href="#directions" title="Directions to
  event" class="directions">
    <div class="aspect" data-ratio="2:1">
      <!-- ... -->
    </div>
  </a>
</div>
```

14. One half and one quarter

The one-half class limits the width of its child elements to 50% and will try and float itself along side other content. It works for siblings like other grid systems, but you also nest them to get quarters.

```
.one-half {
  width: 50%;
  float: left;
}
<!-- Nesting to get quarters! -->
<div class="one-half">
  <div class="one-half">
    </div>
  </div>
```

15. Going responsive

Where we go responsive will depend entirely on the

content you wish to place in the boxes and how you structured the one-half markup. You will have to resize your window and see where things break. In the build example provided there were adjustments needed to the one-half class at 1000px.

```
@media screen and (max-width:1000px) {
  .one-half {width:100%;}
  .one-half .one-half {width:50%;}
}
```

16. 420px and below

We also had to bring the nested halves back to 100% at 420px and below so all the content remained visible. In the build example there are 1:1 boxes inside the nested halves, so we're still able to see two boxes in a portrait view on a mobile device.

```
@media screen and (max-width:420px) {
  .one-half .one-half {width: 100%;}
}
```

17. Changing aspect ratio

There was also a need in the example to modify the aspect ratios as the browser got smaller. This was because they became too large when the halves stacked. It feels counterintuitive to have a ratio of 1:1 look like 2:1, but this is the best performing solution.

```
@media screen and (max-width:1000px) {
  /* Make 1:1 become 2:1 */
  .aspect[data-ratio="1:1"] {
    padding-bottom:50%;}
  /* Make 2:1 become 4:1 */
  .aspect[data-ratio="2:1"] {
    padding-bottom:25%;
  }}
```

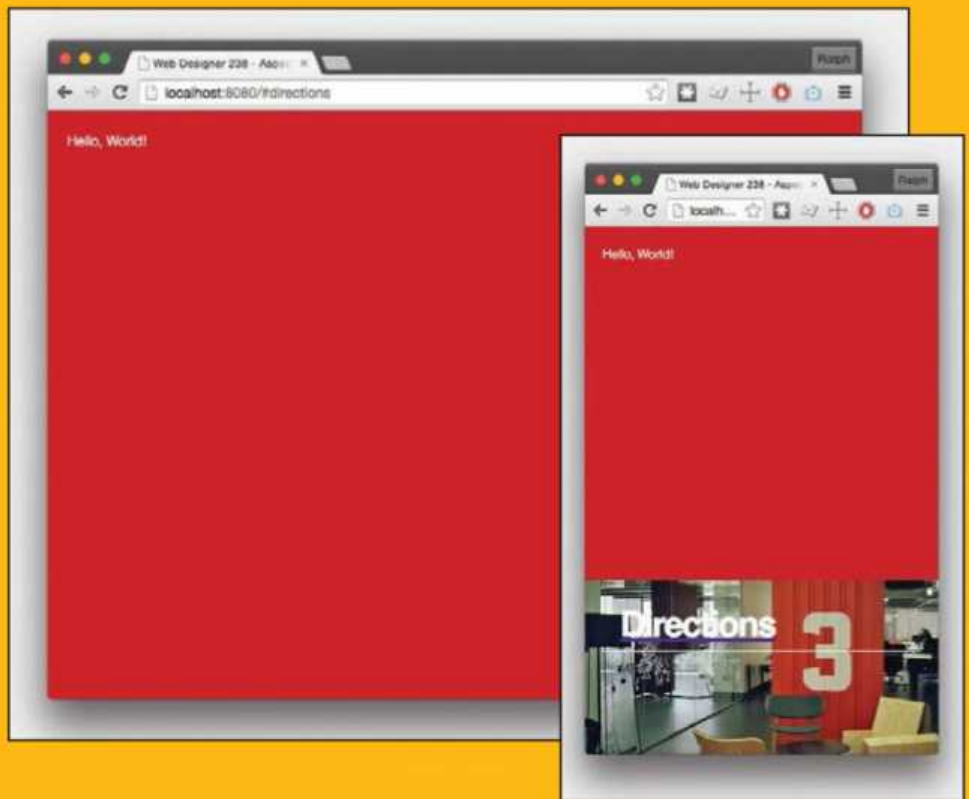
18. Aspect at tablet

The square aspect ratio had a lot of issues in the tablet/large phone area of screen sizes. Similar to the above we've had to take it down to 3:2 at 850px and below and then bring it back to 1:1 at 600px and below.

```
@media screen and (max-width:850px) {
  .aspect[data-ratio="1:1"] { /* Make 1:1
    become 3:2 */
    padding-bottom:66.6%}
  }
  @media screen and (max-width:600px) {
    .aspect[data-ratio="1:1"] { /* Make 1:1
      become 1:1 again */
      padding-bottom:100%}
    }
  }
```

19. Aspect at mobile

At 420px and below, the 2:1 box (which was 4:1 at tablet) was looking a bit thin, even though all the content fit fine with the ratio.



Changing aspect ratio sizes

Unfortunately, aspect ratios don't work across all screen resolutions so when we go responsive we may actually end up with a bad looking UI. There are a few options available to us to fix this problem: write JavaScript to group the elements differently under different .one-half boxes, write JavaScript to modify the data-ratio attribute as required or write CSS to change up the aspect ratios themselves.

These different approaches all have pros and cons. The upside of using this pure CSS solution is that it performs really well across all devices, but it also means that you have elements which say that they are 1:1 but in reality they will have many different aspect ratios as and when the page resizes.

After some trial and error, we found that the 2:1 ratio worked better and it being the original ratio was an added bonus!

20. Finishing touches - markup

Add another call-to-action tile to see how it works with your other tiles in the layout. In the example it sits within a nested one-half container. This is a fairly straightforward CTA as there isn't much content here, it should just work with the responsive styles that have already been written

```
<div class="one-half">
  <a href="#sponsors" title="Sponsors"
    class="sponsors">
    <div class="aspect" data-ratio="1:1">
      <div class="aspect-content">
        <h4 class="headline">Sponsors</h4>
        <p>View our list of sponsors</p>
      </div>
    </div>
  </a>
</div>
```

21. Finishing touches - styles

We're using the .pseudo element to place an image after our tile. Then we absolutely position it. We need to make it clear that these tiles are CTAs on a web platform where users aren't used to seeing them in action.

```
.sponsors .aspect:after {
  content: '';
  width:13px;
  height:20px;
  background-image:url('./img/arrow.png');
  position:absolute;
  right:20px;
  bottom:20px;
  transition: right .5s cubic-bezier(0, 1.14, 1, 1); }
```

22. Finishing touches - animation

To make the hover state a bit more obvious we can change the 'right' property which we set up a transition for in the previous step. It's fairly subtle but reinforces the notion that the user is interacting with an element that will take them somewhere.

Generate sweeping animations with CSS

Make a page-swiper for presentations and the web without overcomplicating JavaScript **tools|tech|trends** HHTML text editor, HTML5 and CSS3 capable browser, HTML, JavaScript and CSS



Advancements in CSS in recent years have delivered an opportunity to reconsider how we approach the management of visual animations that form part of

user experience design. In larger projects that involve specialists in design, programming and content

authoring, it makes sense to keep content, design and functionality completely separate.

The use of CSS to control both design and animation enables designers to take full control of how websites and web-based applications are presented, resulting in the ability to complete visual components faster without distracting programmers from their main project activities. The ability to define and control animations

entirely from within CSS is helpful because it can eliminate the risk of design changes breaking the JavaScript component as all design will have been kept separate to Javascript's functionality code. This tutorial looks at how a swiping page transition effect can be built in CSS, with minimal JavaScript used to detect user interactions and trigger responding animations through the use of CSS.

1. Get started

First declare the main HTML page structure, including the head and body section within the HTML page. Keep styling and functionality separate from the main content by linking from the head section. It is important that browsers know to apply HTML5 standards, so make sure the correct doctype is used.

```
<!DOCTYPE html>
<html>
<head>
<title>Example</title>
<link rel="stylesheet" type="text/css"
href="styles.css" />
<script src="javascript.js"></script>
</head>
<body>
</body>
</html>
```

2. Insert navigation

The content elements are now ready to be inserted. Start with adding the navigation element. This is a HTML5 nav element that acts as a container for links that will show each page when clicked. Using the nav element will be useful later for both styling with CSS and applying interaction listeners through JavaScript.

```
<nav>
<a href="#page1">Page 1</a>
<a href="#page2">Page 2</a>
<a href="#page3">Page 3</a>
<a href="#page4">Page 4</a>>
```

```
<a href="#page5">Page 5</a>
</nav>
```

3. Create content containers

There is need to contain the different groups of content so that we can show and hide them later. Use the article element to create our individual page/screen containers that the different content will be placed in later. Let's make use of a data-status to describe which content section is currently open and visible.

```
<article id="page1" data-status="open">
<h1>Page 1</h1>
</article>
<article id="page2">
<h1>Page 2</h1>
</article>
<article id="page3">
<h1>Page 3</h1>
</article>
<article id="page4">
<h1>Page 4</h1>
</article>
<article id="page5">
<h1>Page 5</h1>
</article>
```

4. Insert the swiper

The swiping effect will be produced by a rectangular element that produce a rotation itself into the screen. Place the element to be animated as part of the HTML. We'll use a div with a unique ID called animElm_swipe for easy reference.

```
<div id="animElm_swipe"></div>
```

5. Create resource files

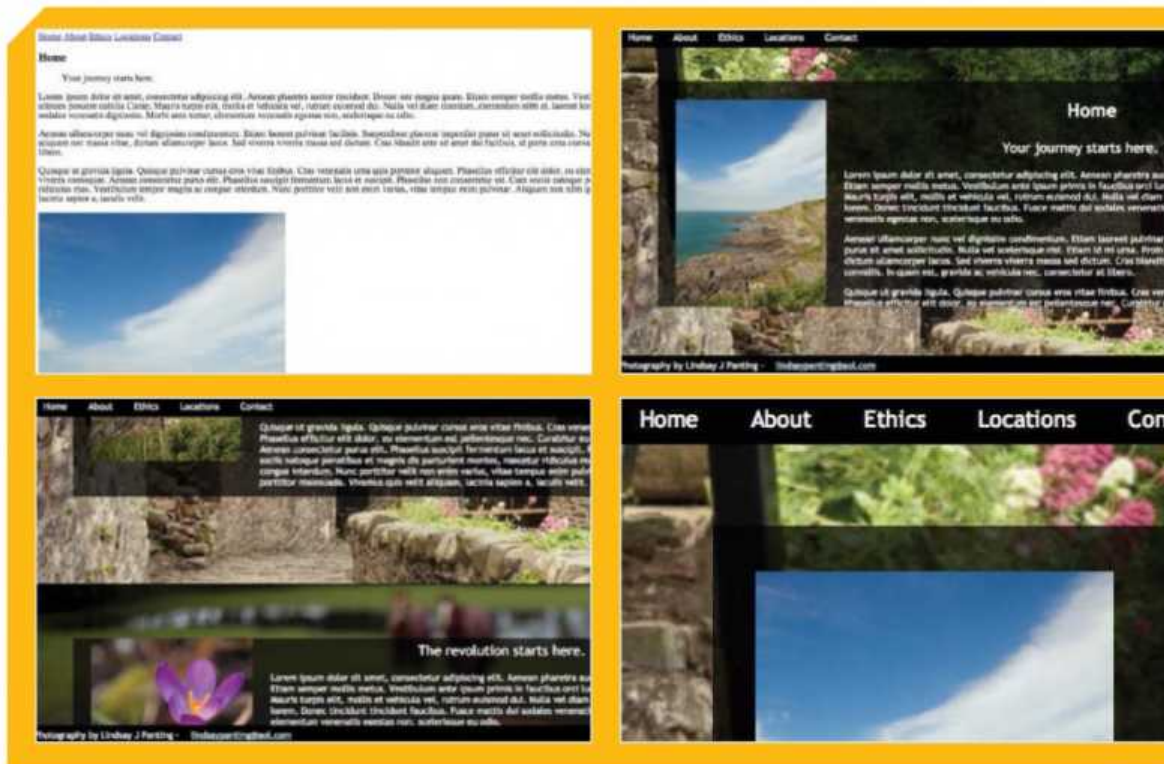
The additional JavaScript and CSS stylesheet resource files are required to add styling and interaction functionality. Create two text files called 'styles.css' and 'javascript.js', making sure that your text editor does not add TXT as a file extension.

```
<html>
<head>
<title>Guided Learning</title>
<link rel="stylesheet" type="text/css"
href="styles.css" />
<script src="javascript.js"></script>
</head>
```

6. Style the articles

Open the styles.css file you have created in your code editor. The first styling to be created will be the articles. Each article represents a screen and will be styled as block elements of full width and length of the screen. Some padding will be used to ensure spacing of content, so the width percentage needs to deduct the percentage used for padding.

```
article{
display: block;
width: 90%;
height: 100%;
clear: both;;
padding: 2em 5% 0 5%;
border-bottom: 1px solid #000;
```



<Top left>

- HTML elements in place with content added to article sections, but no styling yet

<Top right>

- New styles added to the stylesheet now make the article sections and navigation even more presentable

<Bottom left>

- We can still scroll between article sections - we don't have to have this if we want to simulate separate pages

<Bottom right>

- Navigation buttons will be used later to switch pages during swipe animation

```
background: #fff;
}
```

7. Article overflow

We want to adjust the article so that content that doesn't fit onto a single screen can be scrolled without breaking future steps in the tutorial. So set each article to have an overflow of auto, meaning that each article can be scrolled individually without affecting the main screen.

```
article{
display: block;
width: 90%;
height: 100%;
clear: both;
padding: 2em 5% 0 5%;
border-bottom: 1px solid #000;
background: #fff;
overflow: auto;
}
```

8. Fix navigation

The navigation needs to cover the full width and have a relevant height and follow the user wherever they are looking. Attach the navigation to the top-left corner of the web page using fixed positioning so that the coordinates provided using 'top' and 'left' always remain the same and have a z-index on top.

```
nav{
display: block;
position: fixed;
```

```
top: 0;
left: 0;
width:100%;
min-height: 2em;
background: #000;
z-index: 9999;
}
```

9. Navigation links

Links inside the nav element need to be easily selectable for easy access to each page so make each link a block element that has padding. We want each link to appear next to each other, so a display type of inline-block is used to enable padding to be added without forcing each element onto a new line.

```
nav a{
display: inline-block;
color: #fff;
padding: 0.5em 1em 0.5em 1em;
text-decoration: none;
}
```

10. Navigation hover

It makes sense to show the user what they are about to select. Add a hover rule so that the links inside the nav element can be used to change the appearance of the links that are about to be clicked. In this example, we'll change the colour of the text to red.

```
nav a:hover{
color: #c00;
}
```

Style separation benefits

Keeping your visual styling and animations in CSS means that you can keep your JavaScript cleaner and make it easy to update your styles and animations without risking functionality breakage.

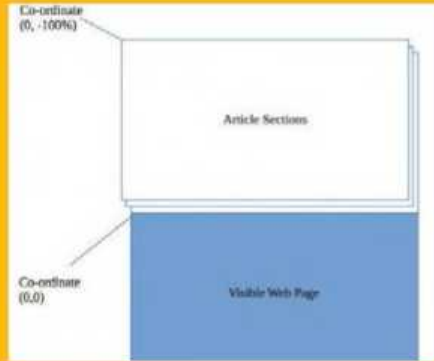
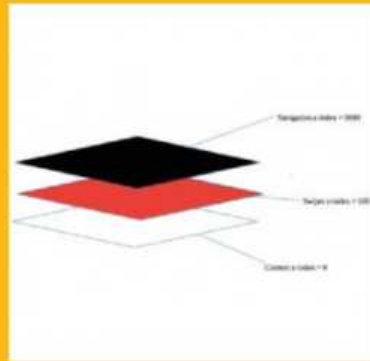
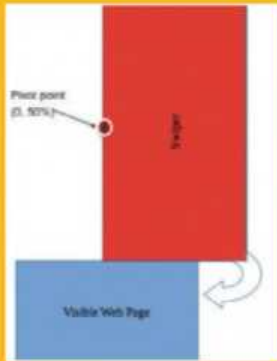
11. Style the swiper

The swiper will be a big box that is rotated over the content to give an illusion that the content is being wiped away. This box needs to be big enough so that no side gaps can be seen when it is being rotated to cover the content. Experiments show that a width of 200% and height of 500% are adequate.

```
#animElm_swipe{
display: block;
width: 200%;
height: 500%;
background: #c00;
}
```

12. Swiper positioning

It is important that the swiper is hidden from view and also easy to rotate when required. Update the swiper to be positioned halfway across the screen and rotated at an angle ready to be rotated into view. Set the transform origin to the top-left corner to make sure the sweep uses the full box as the sweep effect.



<Top left>

- The swiper can now be activated by applying the 'data-animation="in"' or 'data-animation="out"' attribute with ID animElm_swipe

<Top right>

- It's at the 50 per cent point when the swiper covers the full screen data-status="open" switches to visible

<Bottom left>

- The swiper is positioned ready to be rotated over the visible content for wipe illusion

<Bottom middle>

- It is important that the swiper's z-index is between the content articles and the navigation

<Bottom right>

- Article content containers are now adjusted so that they sit above the user's view, enabling them to be dropped into view during transitions

```
#animElm_swipe{
display: block;
position: fixed;
top: -240%;
left: 50%;
-webkit-transform: rotate(-90deg);
-webkit-transform-origin: 0 0;
transform: rotate(-90deg);
transform-origin: 0 0;
width: 200%;
height: 500%;
background: #c00;
}
```

13. Swiper layer

There needs to be guaranteed visibility of the swiper over the content for the effect to work, but it shouldn't appear over the navigation. No z-index has been declared for the article containers and a z-index of 9999 has been used for the nav element, meaning that the swiper can use a z-index between 1 and 9998.

```
#animElm_swipe{
display: block;
position: fixed;
top: -240%;
left: 50%;
-webkit-transform: rotate(-90deg);
-webkit-transform-origin: 0 0;
transform: rotate(-90deg);
transform-origin: 0 0;
width: 200%;
```

```
height: 500%;
background: #c00;
z-index: 100;
}
```

14. Declare swiper animations

There will be two animations for the swiper going from left to right on alternate clicks/taps. Animations work by declaring keyframes and specifying unique keyframes at each percentage. As a minimum, there must be a keyframe at the start (0%) and at the end (100%). Safari and Opera will need a special -webkit version of the keyframes and transform commands.

```
@-webkit-keyframes anim_swipeIn{
0% { -webkit-transform: rotate(-90deg); }
100% { -webkit-transform: rotate(180deg); }
}
@-webkit-keyframes anim_swipeOut{
0% { -webkit-transform: rotate(180deg); }
100% { -webkit-transform: rotate(-90deg); }
}
@keyframes anim_swipeIn{
0% { transform: rotate(-90deg); }
100% { transform: rotate(180deg); }
}
@keyframes anim_swipeOut{
0% { transform: rotate(180deg); }
100% { transform: rotate(-90deg); }
}
```

15. Attach swiper animations

Keyframes only describe animation properties that can

be applied to elements. We want to specify attributes that can be attached to the swiper via JavaScript to trigger animations, enabling us to keep the animation design separate from the functionality code. We are using data-animation attribute set to 'in' or 'out' to trigger a left or right swipe.

```
#animElm_swipe[data-animation="in"]{
-webkit-animation: anim_swipeIn 2s forwards linear;
animation: anim_swipeIn 2s forwards linear;
}
```

```
#animElm_swipe[data-animation="out"]{
-webkit-animation: anim_swipeOut 2s forwards linear;
animation: anim_swipeOut 2s forwards linear;
}
```

16. Position articles

The standard position of each article is shown directly under the previous version and doesn't act like the slideshow we are aiming for. Making the article elements absolutely positioned on a fullscreen height above the visible screen will make sure they are hidden away until we need to show them.

```
article{
position: absolute;
top: -100%;
left: 0;
}
```


17. Article change animation

Now define the timing for the pages to switch in the animation. We define this as an animation that runs synchronised to the main swiper animation, enabling the page to be switched when it covers the full screen.

```
/* ANIMATIONS: Chrome, Safari, Opera */
@-webkit-keyframes anim_pullIn{
  0% { top: -100%; }
  50% { top: -100%; }
  51% { top: 0; }
  100% { top: 0; }
}
@-webkit-keyframes anim_pullOut{
  0% { top: 0; }
  50% { top: 0; }
  51% { top: -100%; }
  100% { top: -100%; }
}
/* ANIMATIONS: Standard syntax */
@keyframes anim_pullIn{
  0% { top: -100%; }
  50% { top: -100%; }
  51% { top: 0; }
  100% { top: 0; }
}
@keyframes anim_pullOut{
  0% { top: 0; }
  50% { top: 0; }
  51% { top: -100%; }
  100% { top: -100%; }
}
```

18. Attach article animations

Although we have defined the animations to use in the page transition, we still need to attach them to the article pages. These animations will only be triggered when specific attributes are attached to the articles, which will be when data-status is set to open or closed.

```
article[data-status="open"]{
  -webkit-animation: anim_pullIn 2s forwards linear;
  animation: anim_pullIn 2s forwards linear;
}
article[data-status="closed"]{
  -webkit-animation: anim_pullOut 2s forwards linear;
  animation: anim_pullOut 2s forwards linear;
}
```

19. Start the JavaScript

The styling and animation definitions are now complete, but no animations are visible because there is no way to trigger the animation attributes we have defined. This can be solved with Javascript - open the javascript.js file and enter the following snippet that enables a shorthand search for elements on the page:

```
//.. helper functions to easily select DOM
```

```
element
var $ = function(cssRule){
  return document.querySelector(cssRule);
}
var $$ = function(cssRule){
  return document.querySelectorAll(cssRule);
}
```

20. Left and right

With the swiper needing to alternate between swiping left and right, there is a need to store the direction to move next. This is achieved by declaring a reference string, which we will call swipeDirection, that will store the direction to use next and will be updated to the next direction when a swipe has been performed.

```
//.. menu click/touch function
var swipeDirection = "left";
```

21. Page change function

The activation of the page swipe and change will be defined in a function called navClick. The function waits until it is called and is not executed straight away, meaning that we can define multiple events that trigger a page swipe and change without duplicating any code. The function accepts a parameter clickElm that contains details of the link clicked. Animations are activated by applying the setAttribute method to set the value of attributes of data-animation and data-status that trigger the animations from the CSS.

```
//.. menu click/touch function
var navClick = function(clickElm){
  //.. set swipe animation
  if(swipeDirection == "left"){
    $("#animElm_swipe").setAttribute("data-
    animation","in");
    swipeDirection = "right";
  }else{
    $("#animElm_swipe").setAttribute("data-
    animation","out");
    swipeDirection = "left";
  }
  //.. set page transition
  $('article[data-status="open"]').
  setAttribute("data-status","closed");
  //.. find the article ID that the link
  refers to: the text after the #
  var articleID = clickElm.href.split("#")
  [1];
  //.. attach data-status="open" to the
  element with ID of articleID
  $("#"+articleID).setAttribute("data-
  status","open");
}
```

22. Apply listeners

Everything is now in place, except for the ability to detect when a navigation link is clicked. This can only be done when the page loads, so this part is put inside

the window's 'load' listener, which triggers code to search for all <a> tags inside the <nav> tag. These elements have a 'click' and 'touchstart' event listener attached to them that call the previously defined navClick function when the user selects them, providing details.

```
//.. window listener - to perform DOM
operations when the page has loaded
window.addEventListener("load", function(){
  //.. get all navigation links
  var navLinks = $$("#nav a");
  //.. loop through all even navigation links
  found
  for(var i=0; i<navLinks.length; i++){
    //.. add click listener to navigation the
    current navigation link
    navLinks[i].addEventListener("click",
    function(e){
      navClick(this);
    });
    //.. add click listener to navigation the
    current navigation link
    navLinks[i].addEventListener("touchstart",
    function(e){
      navClick(this);
    });
  }
});
```

Code Library

JavaScript elements

The JavaScript element kickstarts the animations by listening for navigation interactions that result in the application of animation attributes

```
var $ = function(cssRule){
  return document.querySelector(cssRule);
}
var $$ = function(cssRule){
  return document.
  querySelectorAll(cssRule);
}
```

These are functions that enable jQuery-style access to find the necessary web page elements, but without needing to load jQuery.

```
$("#animElm_swipe").setAttribute("data-
animation","in");
```

An example of how the swiper is activated through Javascript by adding the attribute data-animation="in" to the animElm_swipe element.

HTML & CSS

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Make a screen shrink on scroll

inspiration devstars.com

Hidden menu

The regular menu is hidden behind the burger menu and sits over the entire page on the screen when activated.

Resizing sections

Clicking the dots (right) automatically scrolls the page down to the next section, this fits the browser window exactly.

Shrinking images

The graphical content on the screen animates down in size, giving a pleasing shrink effect as the page scrolls upwards.

Reinforcing the brand

A logo can say a lot about any company and Devstars' logo features a series of stars laid out horizontally. For their showcase content, Devstars uses the shape of the logo as a mask for images of work that they've produced. This not only looks good but also reinforces the brand in the homepage design.

Reverse in size
Returning back through the navigation dots causes the page to slide back up and the graphic content to expand in size.



As a web design and development agency it's very important that you can show off what your skills are; if you can combine samples of your work into an innovative presentation on your site then that's even better. That's exactly what Devstars have done on their site. **Devstars.com** is a full multipage site but use the one-page style website idea for the homepage to show off exactly what they do. The site's homepage has a number of navigation dots that you immediately associate with a slideshow and these contain the latest work offerings, while giving an overview of the company.



<comment>
What our experts think of the site

Making your work the star

"Bright bold colours make this site stand out with just enough negative space to make the logo and site content stand out. The homepage's featured content has an animated effect that draws attention to the care the team put into the small details of their site."

Mark Shufflebottom, professor of Interaction Design, Sheridan College

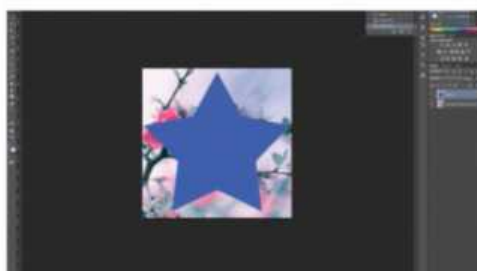
Technique

1. Create the shrinking logo effect

When making this effect work, the logo needs to be placed within a page. As there is no page content here, some div tags are being added, which will have 600 pixels of height added to them. When the image reaches the top of the page, it will be shrunk down.

```
<div class="space"></div>

<div class="space"></div>
<div class="space"></div>
```



2. Add the library

In the head section of the page, the script tag calls the jQuery library so that we can detect the position of the page and make DOM changes based on that. The CSS styling gives the 'space' class a height of 600 pixels.

```
<script type="text/javascript" src="js/jquery.js"></script>
<style>
.space{
height:600px;
/*Purely Demonstration Purposes*/
}
```

3. Finish the style

The image with the ID of 'shrink' will be scaled down when it hits the top of the page - this shrink will then be detected through jQuery code later. The image is displayed as a block element, centring it horizontally.

```
#shrink{
display: block;
margin: 0 auto;
}
</style>
```

4. Detect the position

Just before the closing body tag, the script tag is added along with any of the code that follows. This script tag will check how far down the page the document has actually scrolled.

```
<script >
$(function () {
$(window).on('scroll', function() {
var scrollTop = $(this).scrollTop();
var topDistance = $('#shrink').offset().top;
```

5. Start the shrink

When the image is within the space of the top of the page, the 'shrink' image is animated down to 0 on the width and height over half a second. Now save the page, then you can test the effect in the browser to see the effect in action as you scroll the image to the top.

```
if ( topDistance < scrollTop ) {
$('#shrink').animate({"width": 0, "height":
0}, 500 );
}
});
});
</script>
```

Varied navigation
The homepage combines a one-page style site and a slide show. The scrollbar is hidden but navigation dots are still visible.

Produce an animated off-screen 3D menu

Get your menu to stand out with a 45-degree page view off screen **tools|tech|trends** Brackets



lot of the projects that are featured in this magazine are done so because they have some unique feature about them. Most of these have some

quirky way of interacting with the menu and as the user tends to do most of the interaction with the menu, it's usually a good place to

start when creating a unique focal point for your site. In this tutorial we are going to look at CSS3 transformations that, by default, are hardware accelerated. As such, we can add lovely 3D effects and transitions to our content by adding and removing classes that will trigger the animations.

In this menu we will have an off-screen menu that slides in from the left-hand side, nothing out of the

ordinary about that, except that our page content will flip out of the way using a 3D transform enabling the menu to take full focus on the page. Then, when it's time to bring the page back, the user simply clicks on the page and the menu slides back out and the page rotates back into view. A little trick here to make this work is to stop the page being scrollable while the page is rotated out to one side.

1. Start the project

Open the start folder in Brackets or place it in your local web server folder. Take a look at the page, before starting the project, in a web browser to see that there is a basic page on display – a menu needs to be added to this. There is a comment in the index.html page showing the end of the twist div, add the menu in here.

```
<nav class="offscreen-nav">
<a href="#">Home</a>
<a href="#">News</a>
<a href="#">Blog</a>
<a href="#">Portfolio</a>
<a href="#">Contact</a>
<a href="#">About</a>
</nav>
```

2. Switch to the CSS

Save the index page and move to the style.css in the CSS folder. Add the following code in here. It can go at the bottom of the document, just make sure that it isn't inside a media query.

Here the twist class is being given a relative position on the page. This will hold the menu outside of the page.

```
.twist {
position: relative;
}
```

3. Build the content

All of the real pages go inside the container class. Here it

is given a white background because later the menu will be given a red background to match the design on the screen. It's given a z-index that is higher than the rest of the menu so that all the main content will be visible above this.

```
.container {
background: #fff;
min-height: 100%;
position: relative;
outline: 1px solid rgba(0,0,0,0);
z-index: 10;
-webkit-transform: translateZ(0);
translateX(0) rotateY(0deg);
transform: translateZ(0) translateX(0) rotateY(0deg);
}
```

4. More page positioning

There is a wrapper class just inside the container, again this needs to be set to relative so that when the design opens, it works correctly. When the menu is opened a class gets added to it and this is called 'open'. Here the twist class is made to be fixed and the perspective added.

```
.wrapper {
position: relative;
}
.twist.open {
position: fixed;
-webkit-perspective: 1500px;
perspective: 1500px;
}
```

5. Tidy up the open page

When the menu opens, the container that holds the regular page content is made to have no overflow. This helps it to twist out with a 3D perspective to it without having the rest of the page on display. At this point the cursor is set to be a pointer so that the container becomes the button to bring the page back into the main view.

```
.open .container {
position: absolute;
overflow: hidden;

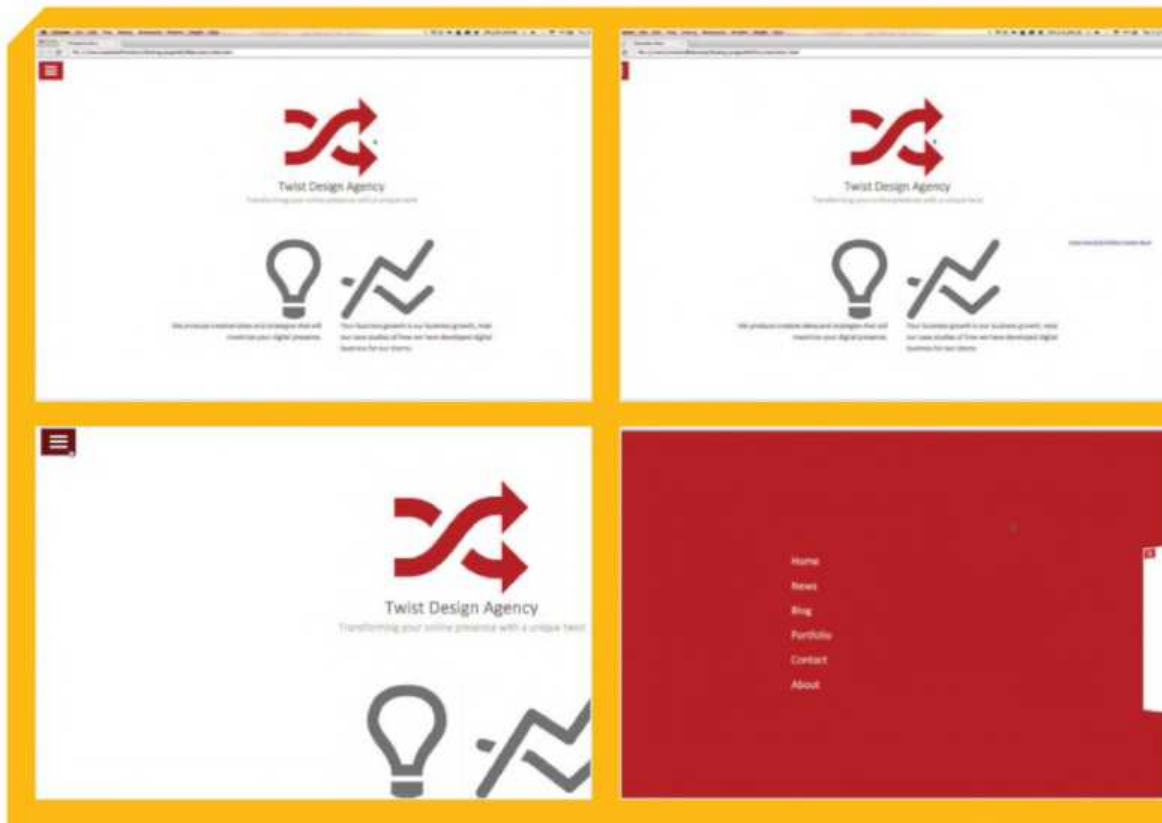
width: 100%;
height: 100%;
cursor: pointer;
-webkit-backface-visibility: hidden;

backface-visibility: hidden;
}
```

6. Add functionality

As the content is being animated in 3D space the open wrapper is given a CSS transformation on the z axis. The container is slightly altered when it is in animation to be slightly bigger than the screen with full opacity. The transition takes less than a third of a second.

```
.open .wrapper {
-webkit-transform: translateZ(-1px);
}
.animate .container::after {
```

**<Top left>**

- The basic page layout is already set up, the tutorial concentrates on adding the menu elements and functionality of this

<Top right>

- The menu is added and immediately appears on the screen, however it isn't styled up yet or set in the right position

<Bottom left>

- The menu is controlled by the icon in the top left-hand corner. The burger menu is later given the functionality to bring out the menu and cause the page to rotate out

<Bottom right>

- The menu is styled in white text and brought onto the screen from the left

```
opacity: 1;
height: 101%;
-webkit-transition: opacity 0.3s;
transition: opacity 0.3s;
}
```

7. Position the navigation menu

Here the navigation for the off-screen menu is set in the CSS. The position is set at absolute so that it can be animated from the side and made to have a height that fits its content. This is given the transform position of 50% of the height and to ensure hardware acceleration the preserve-3d is set.

```
.offscreen-nav {
position: absolute;
height: auto;
font-size: 2em;
top: 50%;
-webkit-transform: translateY(-50%);
transform: translateY(-50%);
-webkit-transform-style: preserve-3d;
transform-style: preserve-3d;
left: 25%;
}
```

8. Menu items

Each menu element needs to be styled up in the right font weight, with the underline and the margin taken off to make it appear in the right place on the screen.

A transition is added so that the text colour can change on rollover and to make it all visible.

```
.offscreen-nav a {
display: inline-block;
white-space: nowrap;
font-weight: 300;
text-decoration: none;
margin: 0 0 30px 0;
color: #fff;
-webkit-transition: color 0.3s;
transition: color 0.3s;
-webkit-transform-style: preserve-3d;
transform-style: preserve-3d;
}
```

9. Finish the menu items

The next CSS will give the hover a bright yellow colour to make it stand out against the red background. The background is set in the effect-persp class to a red colour to match the logo in the page. The white page will rotate out to the right while the menu will animate in from the left on the red background.

```
.offscreen-nav a:hover {
color: #ffff72f;
}
.offscreen-nav a {
display: block;
}
.effect-persp {
```

Normalize.css file

The project makes use of normalize.css, which makes browsers rendering all elements more consistently and in-line with modern standards. It's available from necolas.github.io/normalize.css.

```
background: #b40000;
}
```

10. Add the transition

In order to make the container swing out and twist in 3D space we need to change its transform origin point to the centre of it. This is given a slightly longer transition time than in Step 6, but it all works together to bring the effect on the screen.

```
.effect-persp .container {
-webkit-transition: -webkit-transform 0.4s;
transition: transform 0.4s;
-webkit-transform-origin: 50% 50%;
transform-origin: 50% 50%;
}
```

11. Close the row

When the container is swinging out, a class of 'animate' will be added to it. This is the class that will actually contain the animation. As you can see it rotates the interface on the y axis by 45 degrees. Try and imagine

HTML & CSS



<Top left>

- The rollover for the menu elements is set to yellow to bring a transition highlight colour to the interface

<Top right>

- The CSS controls the page rotating on the y axis so that it moves slightly over to the right and rotates 45 degrees

<Bottom left>

- The menu is in midtransition as the page rotates out of the way and the menu fades in from the left of the browser window

<Bottom right>

- When the menu is open, the page that is rotated becomes the link to bring this back to the forefront. Notice that scroll is disabled in the menu view



Project add-ons

The project uses classie.js, a lightweight class editor that saves using the whole of jQuery, it's available from github.com/desandro/classie.

that there is a pin in the top of the screen that rotates the page 45 degrees away from the view.

```
.effect-persp.animate .container {
  -webkit-transform: translateZ(-1500px)
  translateX(100%) rotateY(-45deg);
  transform: translateZ(-1500px)
  translateX(100%) rotateY(-45deg);
}
```

12. Second row of columns

The actual links are positioned off the screen to the left so the translateX CSS transform is applied to keep them 150 pixels off the screen to the left. These are animated in from a transparent opacity.

```
.effect-persp .offscreen-nav a {
  opacity: 0;
  -webkit-transform: translateX(-150px);
  transform: translateX(-150px);
  -webkit-transition: -webkit-transform 0.4s,
  opacity 0.4s;
  transition: transform 0.4s, opacity 0.4s;
}
```

13. Final CSS

Finally the menu is brought onto the screen with the full

opacity for each menu element. Save the style.css now because it has been completed. There won't be anything to see in the browser though because there is no functionality added to the page yet. That will come next by applying the CSS classes with JavaScript.

```
.effect-persp.animate .offscreen-nav a {
  opacity: 1;
  -webkit-transform: translateX(0);
  transform: translateX(0);
}
```

14. Start the JavaScript

Open the file twist.js and you will see that it is an empty document ready for us to begin. This tutorial is using the classie.js external library for adding and removing classes with JavaScript to CSS. In this function the code returns how much the page has scrolled.

```
function scrolly() {
  return window.pageYOffset || docElem.
  scrollTop;
}
```

15. Setting variables

The next part of the code sets out some variables that are needed in the code. The biggest section is an object containing the browser prefix names. These are used to check when the transition has ended by dynamically adding an event listener to the transition later in the code.

```
var docElem = window.document.
  documentElement,
```

```
support = "transition",
transEndEventNames = {
  'WebkitTransition': 'webkitTransitionEnd',
  'MozTransition': 'transitionend',
  'OTransition': 'oTransitionEnd',
  'msTransition': 'MSTransitionEnd',
  'transition': 'transitionend'
},
transEndEventName = transEndEventNames[
  'transition' ],
docscroll = 0;
```

16. Initialise the interface

Most applications have an init function to initialise all the things that are necessary. The first part of this function will get a reference to all the necessary elements in the DOM so that these can be manipulated through the code without having to continuously traverse the DOM.

```
function init() {
  var showMenu = document.getElementById(
    'showMenu' ),
  twistWrapper = document.getElementById(
    'twist' ),
  container = twistWrapper.querySelector(
    '.container' ),
  contentWrapper = container.querySelector(
    '.wrapper' );
}
```

17. Show the menu

Here the menu button is detected for when a user clicks on it. The click event fires the remaining function, which is only partially shown in this step. The event is stopped



<Above>

- The final design works on a variety of different screen sizes, with the menu being placed in the space available

from propagating and the default action of the button is also prevented. This enables the code to run without default actions interfering.

```
showMenu.addEventListener( 'click',
function( ev ) {
ev.stopPropagation();
ev.preventDefault();
docscroll = scrollY();
```

18. Finish the image

The rest of the function is shown here. The scrolling is set to stop at this point as the class of 'open' is set to the twistWrapper element. Just marginally after this, triggered by the setTimeout command, another class is added called the 'animate' class and this therefore starts the animation.

```
contentWrapper.style.top = docscroll * -1 +
'px';
document.body.scrollTop = document.
documentElement.scrollTop = 0;
classie.add( twistWrapper, 'open' );
setTimeout( function() { classie.add(
twistWrapper, 'animate' ); }, 25 );
});
```

19. Return the menu

The container gets an event listener added to it, which also detects input from a click. This should only fire if it has the 'animate' class already added to it because that means it's actually open and needs to go back when clicked, otherwise it won't fire.



Integrate the menu

This menu could easily be implemented with a responsive framework such as Bootstrap or Foundation as an alternative to their respective menus. Remember that each of those frameworks give you a custom download of what is available, you don't need the whole framework. As such, you can leave out the existing menu that comes with those frameworks

and add your own. Doing this is a great way of customising and making the design look less like an existing framework because those menus are very obvious. Custom web design menus will always create a more appropriate look for your web projects and with so many responsive frameworks, it's almost possible to take the best parts of each to kick-start your own work.

```
container.addEventListener( 'click',
function( ev ) {
if( classie.has( twistWrapper, 'animate' ) )

{
var onEndTransFn = function( ev ) {
if( support && ( ev.target.className !==
'container' || ev.propertyName.indexOf(
'transform' ) == -1 ) ) return;
```

20. Remove the event

If the transitions have finished then the 'open' class needs to be removed, which makes sense as the menu isn't open any more.

At this point scrolling is returned to the user so that the page can carry on like a normal web page under the control of the user.

```
this.removeEventListener(
transEndEventName, onEndTransFn );
classie.remove( twistWrapper, 'open' );
document.body.scrollTop = document.
documentElement.scrollTop = docscroll;
contentWrapper.style.top = '0px';
};
```

21. Removing the animate class

Similar to the previous step the twistWrapper has an event listener added that detects that the transition has finished. When it does it removes the class of 'animate' as this is no longer applicable to it. Just the final finishing off of the init function is left.

```
twistWrapper.addEventListener(
transEndEventName, onEndTransFn );
classie.remove( twistWrapper, 'animate' );
}
});
```

22. Finish off

Add the final code and bracket to close off the init function. The final line calls this function so that the previous code is applied. Now save this JavaScript and open the index.html page in your browser to see the menu open and close while the transition takes full effect.

```
twistWrapper.addEventListener( 'click',
function( ev ) { return false; } );
}
init();
```

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Slide up titles on page load using CSS

inspiration onedollarlesson.com

Sobering statistics

The animations deliver some frightening stats about the levels of cybercrime targeting user's bank details.

Well-laid out menu

The menu is neatly tucked away to the left. As the user scrolls the menu stretches to indicate position on page.

Claim your prize

Complete all three of the lessons to access a free three-month Kaspersky security trial. A very useful prize.

Content delivery

Packaging important information into an entertaining, easily consumable format has become one of the primary challenges for websites, especially when delivering information that has not necessarily been sought out. One Dollar Lesson employs just the right mix of interaction and spectacle.

Hidden animations

Scrolling past the first screen will reveal some beautifully crafted animations. These unfold as the users scrolls down.

On page load

Once the initial load animation (featuring a dollar) is done, the title elements begin their step-by-step animation.

So you're looking for a good way to engage your website users from the moment the homepage loads. Grabbing the diminishing attention spans of web users is an on-going battle for developers. There are all sorts of options, from autoplay videos to pop-ups which demand user interaction. Sometimes, though, you can keep the user's attention for those first few vital seconds with a simple animation. Nothing too over the top, just a little something that keeps them watching.

How about an animation that delivers the site title in steps, over a short period of time?

Internet security company Kaspersky have put together this wonderful-looking microsite, designed to educate users on the dangers of online payments and the many related cyber attacks that users can fall prey to. Featuring some great scroll-activated animations and fullscreen videos (worth checking out for the transport-based demonstrations of Trojans and the like), One Dollar Lesson opens with a perfect example of the effect we are going to re-create in this workshop. Once the homepage background has loaded, the site title and the Kaspersky logo ease into view, one at a time. Simple effect, but one that keeps you watching.



<comment>
What our experts think of the site

Make the wait interesting

"These days it's a brave website that asks the user to sit through a content-loading percentage bar, however short the wait. The desire for instant content is relentless. If this is unavoidable you better make sure your loading icons are interesting and different enough to hold users fast."

Richard Lamb, owner and web designer at Inspired Lamb Design

Technique

1. Background image

First, put in place the fullscreen background image. We've selected an image which places content focus on the left, so that we can place our animated content on the right. Ensure that you include your vendor prefixes.

2. The base HTML

Create an intro div that will sit on the left side of the canvas (using Bootstrap CSS makes it simpler). Within this we need four title divs, each containing a div class we're calling yum. The yum classes will contain the animations.

```
<div class="intro col-md-3 col-md-push-8">
  <div class="title">
    <div class="yum"></div>
  </div>
  <div class="title">
    <div class="yum"></div>
  </div>
  <div class="title">
    <div class="yum"></div>
  </div>
  <div class="title">
    <div class="yum"></div>
  </div>
</div>
```

3. The initial CSS

The intro div should be given some slight padding at the top. The title div needs to assign an overflow: hidden property. We need the child elements invisible until they reach the confines of their parent. We'll call this 'rise'.

4. Insert the title images

Create four PNG images for the three words and an underline, of equal width, which make up our title, 'Um

Num Num!'. Place one into the four yums. Give each of div an individual class name, to match the image names.

5. Individual animation times

Then, set animation durations specific to each of our individual divs.

```
.um {
  animation-duration: 2s;
}
.num-one {
  animation-duration: 4s;
}
.num-two {
  animation-duration: 6s;
}
.underline {
  animation-duration: 7s;
}
```

6. Write the keyframes

The keyframes re-creates the effect from our inspiration site causing the four title elements to slide up from the bottom, with a slight bounce on arrival.

```
@keyframes rise {
  0%, 60%, 75%, 90%, 100% {
    transition-timing-function: cubic-bezier(0.215, 0.610, 0.355, 1.000);
  }
  0% {opacity: 0; transform: translate3d(0, 3000px, 0);}
  60% {opacity: 1; transform: translate3d(0, -20px, 0);}
  75% {transform: translate3d(0, 10px, 0);}
  90% {transform: translate3d(0, -5px, 0);}
  100% {transform: translate3d(0, 0, 0);}
}
```


↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Animate scroll-reveal split backgrounds

inspiration 2015.extralagence.com

The advent of responsive web design has gradually led to an increasing popularity in single page sites.

Scrolling is now considered to be as user-friendly as clicking and the invention of more and more scroll effects to enhance the experience has given web designers some wonderful toys to play with.

Working with backgrounds while scrolling has given us parallax effects, and there are jQuery functions for changing the background colour while scrolling. But how about splitting and halving your background so that both sides change as the user scrolls?

Sounds like a disaster but, it can actually deliver a striking and unique backdrop. You can check out the files for this tutorial on FileSilo.



<comment>
What our experts think of the site

Don't distract

"Sometimes even the most impressive animation can be accomplished in its execution but detrimental to the project as a whole. The scroll effect in Extra's website is one of those that, if not handled well, could act as unnecessary distraction. With sparse, image-based content this effect works, but with heavily text-based content it could be counterproductive."

Richard Lamb

The other site

It is well worth visiting Extra's other site at extralagence.com to see another example of left/right sectioning and layout from the agency. This is more complex, with slightly different content in each. Notice how the edge on each side moves left and right.

Text behind figures

There is a cool effect on the *Fanatiques* and *Illuminés* section where one of the staff members appears to be standing in front of text. It's a simple cropping trick.

Varied navigation

The site offers two forms of navigation through the sections; you can use the arrow icon at the bottom of the page, or the dot navigation on the left side.

Megaphone motif

Check out the custom loading icon when the site first loads. It feels like the agency has something to say and it's based on the megaphone from another section.

Active imagery

The animated images, which are central to each section, are achieved by switching the visibility of two images, each taken at a slightly different angle.

Mobile responsive

Take a look at the site on your mobile to see how each section is laid out differently, depending on the content and image. The animated backgrounds and bright colours remain, however.

Inspiration Something Extra

The Extra Agency is a French digital agency specialising in communications and branding for the sports industry. They have a number of websites and blogs, but the focus of this web workshop is an effect found on their 2015 side site.

Technique Combine the effects

1. Set the base HTML

The base HTML will consist of three sections, each representing our three 'pages'. Each section should be given a unique ID and a binding class. Within each section we need to create two parent classes of container and background, which will be layered with the CSS. Place your content in the container and set unique left and right divs in the background.

```
<section id="one" class="screen">
  <div class="container">
    <div class="image-place">
      <h1>American Rail</h1>
      
    </div>
  </div>
  <div class="background">
    <div id="one-left"></div>
    <div id="one-right"></div>
  </div>
</section>
```

2. Set the base CSS

You can set whatever width to your container that you want and make it responsive. The final effect will not be altered. The binding class added to each section and screen should be given a relative position and 100% width. Add a minimum height if you wish.

```
.container {
  margin:auto;
}
.screen {
  position: relative;
  display: block;
  width: 100%;
  overflow: hidden;
  min-height: 360px;
}
```

3. Background CSS

We need to set a default colour for our background div, and give it an absolute position and a low z-index, so it can sit behind the relatively positioned content to come. Each uniquely named left and right div should also be given the same position style and their widths and height

This one-page website features some notable effects, including a neat split-background effect. Scroll down beyond the initial landing section and you will see that the background for each subsequent section animates in halves, one side easing down and the other easing up. The result is an intriguing effect, one we're going to re-create, utilising the Scroll Reveal JavaScript plugin.

in percentages as in the code.

4. Choose colours

Now you can set the background colours for each of the left and right divs in each section. The choice is entirely yours, but bear in mind that too striking a contrast in your halves will prove very distracting to the user. Harmonious colours work well or two shades of the same colour is also a good idea.

```
#one-left {
  background-color: #7c2323;
}
#one-right {
  background-color: #9d4242;
}
#two-left {
  background-color: #385f7d;
}
#two-right {
  background-color: #5d84a1;
}
#three-left {
  background-color: #416e2f;
}
#three-right {
  background-color: #5c844c;
}
```

5. Prepare the content

Before we can set about arranging the animation, we need to take care of our content. Any padding and alignment will depend on your content, but you do need to ensure that there is a declared position and a z-index which will bring the content in front of the background. If you are working responsively, your styles should reflect this.

```
.image-place {
  padding: 100px 0 130px;
  position: relative;

  z-index: 10;
  max-width: 100%;
  text-align:center;
}
.image-place img {
  width:600px;
  border:10px solid #eee;
}
```

Technique Backdrop animation

In order to begin animating our background halves, we'll need to visit github.com/julianloyd/scrollReveal.js and download scrollReveal.min.js. Place it in your root folder. Once in place, implementing the functions is a relatively simple process.

1. Initialise ScrollReveal

Remember to link to the jQuery library before adding the link to scrollReveal.min.js. Add the short script for Scroll Reveal functions so you can investigate the keywords of the element animations.

```
<script src="https://ajax.googleapis.
com/ajax/libs/jquery/1.11.2/jquery.
min.js"></script>
<script src="scrollReveal.min.js"></
script>
<script>
  window.sr = new scrollReveal();
</script>
```

2. Declare the animations

Scroll Reveal offers a comprehensive list of keywords that can be entered into a data-sr declaration for each element. These initiate animations once the element is scrolled into and, if you wish, out of view. Check the GitHub link for the full list.

```
<div id="one-left" data-sr="enter top
move 500px reset"></div>
<div id="one-right" data-sr="enter
bottom move 500px reset"></div>
```

3. Breaking it down

'Enter left' declares the direction in which the element should slide into the window. 'Move 500px' tells the element the distance to travel between starting point and end point. 'Reset' makes the animation reverse when elements scroll out of view.

```
<div id="three-left" data-sr="enter left
move 500px reset"></div>
<div id="three-right" data-sr="enter
right move 500px reset"></div>
```

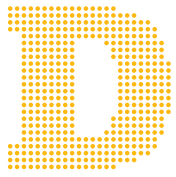
🔍 Before we can arrange the animation, we need to take care of our content 🗨

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Animate an information card box with CSS

Create an expanding card-like box with changing text on hover





esign sometimes creates a requirement for information to be presented in a restricted amount of space. This problem has traditionally been solved by reducing the amount of

information, which compromises the purpose that the design seeks to serve. With websites and apps enabling interactivity to be integrated within design, there are opportunities to solve this problem so it doesn't compromise design or content.

Information cards are a method for websites and apps to present information on a single screen in a way that can be expanded through interaction. These elements can contain information as well as features such as navigation, forms and diagrams that the default design doesn't provide enough room to present. In addition to the spacing issue, information cards can be used to make a design appear to be less cluttered when there is a lot of information to present. This tutorial shows how information cards can be used for a navigation component that expands to show a preview of the pages being linked to upon being hovered by the mouse cursor.

1. Get started

First, declare the main HTML page structure including the head and body section within the HTML page.

2. Content container

There is a need to contain the page content in some type of container so that we can automatically control the flow of the page element such as for adapting the page to different screen resolutions. We will create this using a <div> element with an ID called 'container' that can be directly referenced from the CSS.

3. Content overview

With the main page foundations in place, we are now ready to start inserting the main content. The first elements to add inside the container will provide an introduction to the page content - these being the page title page and summary. This will provide the information needed by the reader to identify whether they want to investigate further.

4. Card navigation

The information cards are being used as navigation options, so the next step is to define each card as a container for their content in a way that can open a webpage when clicked. This is achieved by using <a> tags within a <nav> container, hence the cardset being treated as any HTML regular navigation. Place the following code inside the content container:

```
<nav>
<a href="http://www.google.com">
</a>
<a href="http://www.google.com">
</a>
<a href="http://www.google.com">
</a>
<a href="http://www.google.com">
</a>
<a href="http://www.google.com">
</a>
</nav>
```

5. Card content

With each of the <a> tags being used as the card container, we now need to insert the content to be displayed as part of the cards. This content will contain

an image and two types of text - one for the introduction and another more detailed set of text to be shown when the information card is being hovered over.

```

<span>
Sun
</span>
Intro text...
</span>
<span>
More text to be shown on hover...
</span>
</span>
```

6. Initiate CSS styling

Create a text file called 'styles.css', making sure that your text editor does not add TXT as a file extension. These also need attaching from the <head> section of HTML document.

```
<link rel="stylesheet" type="text/css"
href="styles.css" />
```

7. Style HTML body

It is important not to forget that the <html> and <body> elements on a webpage are set to a height of just one

HTML with selectors

There is no need to complicate your HTML and CSS with class names, IDs and properties when you can reference elements using their ordered position, with selectors such as :nth-child().



<Left>

- Default information card styling now positions each of our cards horizontally

<Top left>

- HTML are now elements in place, with content added to article sections, but there is no styling yet

<Top right>

- The cards have been updated with styling to show the page container with the information card content

HTML & CSS

line by default, hence causing problems if we want to create a container that has a full height of the webpage. We solve this by setting the `<html>` and `<body>` elements to be full height in the CSS.

```
html,body{
display: block;
width: 100%;
height: 100%;
margin: 0;
padding: 0;
}
```

8. Page container

The container needs to be styled in a way that it will stand out. This can be done by changing the main page background colour and positioning the container in the centre of the page. To do this, we need to update the `<head>`, `<body>` and container sections.

```
html,body{
font-family: "Trebuchet MS", Helvetica,
sans-serif;
background: rgb(21, 34, 47);
}
#container{
display: block;
margin: 0 auto 0 auto;
padding: 1em;
```

Foregoing pages

Making use of information cards instead of making the user switch between pages is helpful as it simplifies code - it avoids the need of complicated JavaScript to save temporary data.

```
background: #ccc;
width: 1000px;
height: 100%;
}
```

9. Card container

In addition to the page having a container, we are also using `<nav>` as the container for the information cards. This container should appear to separate the cards from any other page content, as well as to define the height and width used to contain the cards. Displaying the `<nav>` as a block element will automatically make it stretch the full width of the page container.

```
nav{
position: relative;
display: block;
padding: 1em;
border: 3px solid #000;
margin: 2em 0 2em 0;
height: 80%;
overflow: hidden;
}
```

10. Define cards

The information cards need to be defined in relation to the `<nav>` container they are placed within. This step defines their default width, height and colour that they will be displayed. These elements will be displayed as inline-block elements so that they can be positioned next to each other at a width that fits all five items.

```
nav a{
display: inline-block;
padding: 0;
font-size: 3em;
text-decoration: none;
```

```
width: 18.5%;
height: 99%;
border: 3px solid #fff;
background: rgb(81, 173, 228);
color: #fff;
overflow: hidden;
}
```

11. Card transition

It is important that the cards have transitions enabled so that they can appear to animate without the need for JavaScript. We will set all properties of the card containers to have a transition of one second, enabling us to animate all properties from colour to size.

```
nav a{
-webkit-transition: all 1s;
transition: all 1s;
}
```

12. Card open animation

Activation of animations is now just a case of defining the properties to animate to. This is achieved by using the `:hover` attribute on the `<a>` tag of the `<nav>` element, which sets the animation to only occur when the information card is being hovered over. The main animation property will be setting the information card to cover half the width of the container.

```
nav a:hover{
color: #333;
width: 50%;
margin-top: 0;
}
```

13. Adapt other cards

The problem with opening the current hovered card to



<Top left>

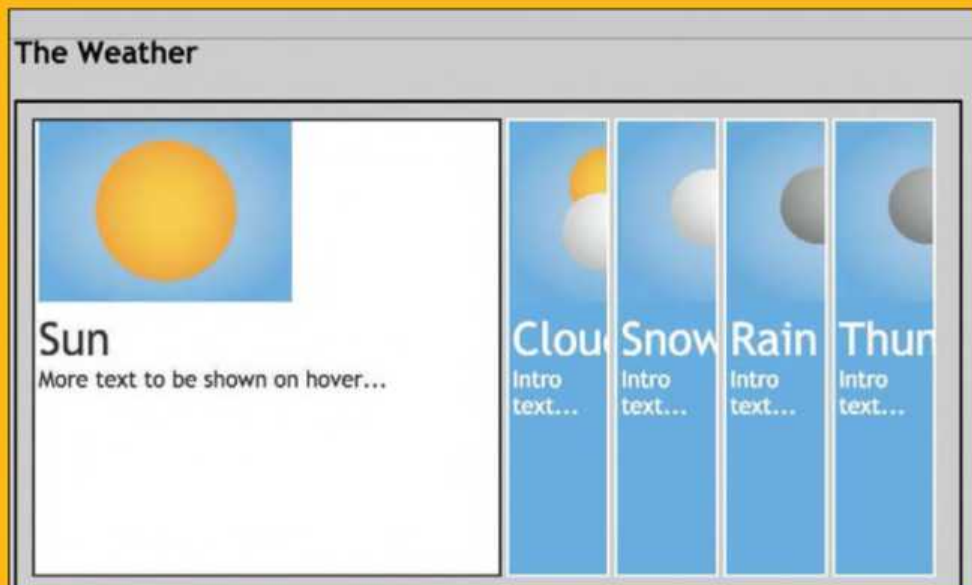
- Here we've reduced the width of the information cards when their nav container is hovered

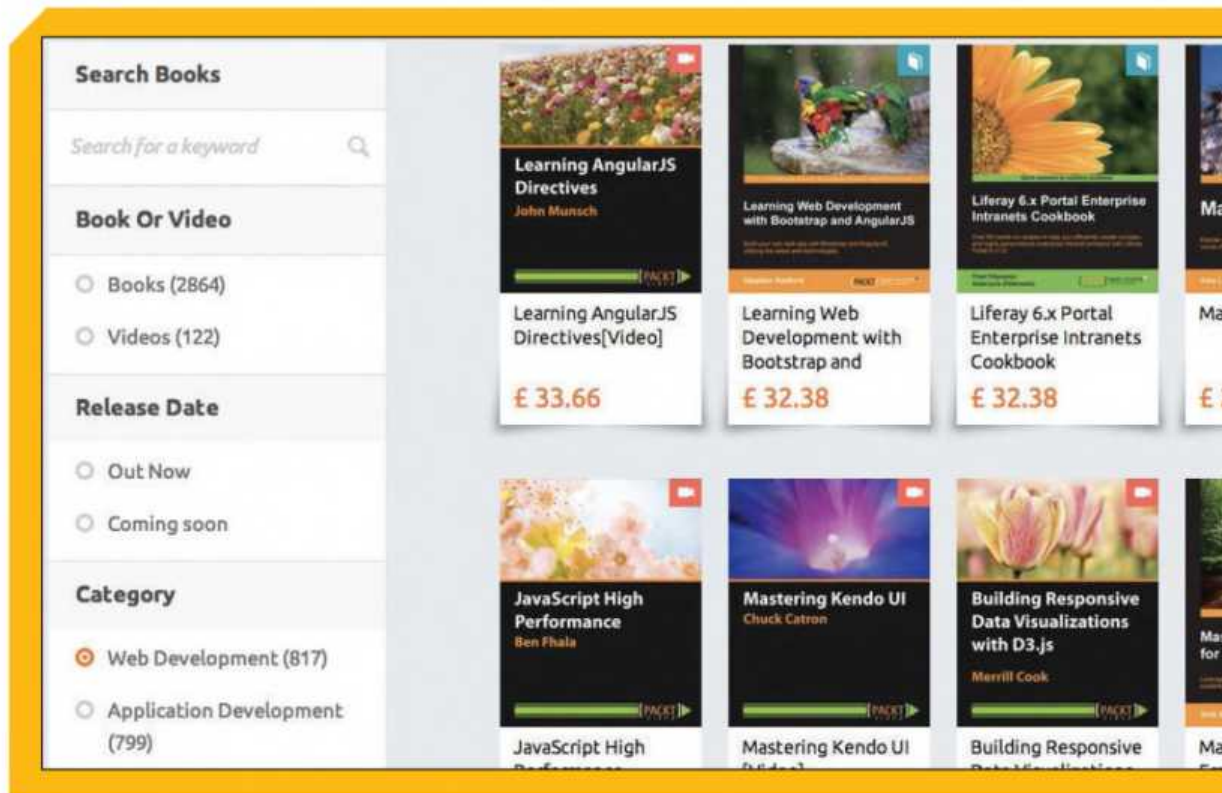
<Top right>

- The text size of the inner `` elements is now half the size of the standard heading text

<Right>

- Adding a white background to the hovered information card makes it stand out more, but the picture icon looks out of place





Use cards for apps and eCommerce

You can utilise information cards to extend product listings for an eCommerce website or as part of an online information brochure. The ability to present more information without the need to reload the webpage will provide more convenience to website visitors and will result in them being more willing to look at more information on the page. Better visitor engagement will enable your website to produce better commercial results, whether it be increased sales conversions, enquiries or other actions that your content is designed to produce. Aside from website-oriented projects, information cards can be used for web-based apps, enabling users to access different features without having to switch pages.

50% of the screen is that it will push other cards out of view. This can be solved by animating the other cards to have a smaller width when the <nav> container is hovered so it's activated at the same time as the card.

14. Content sections

The elements inside of the information cards are used instead of <div> elements to keep the HTML markup compliant with W3C standards. We can use CSS to make these elements act in the same way as <div> elements by setting the display property to 'block', enabling them to have their size and padding set in addition to placing each element on a new line.

```
nav a span{
  display: block;
}
```

15. Preview content

The structure of the HTML uses a tag to contain the card information, with two more tags inside this to contain the preview and full content. We can use CSS to define consistent properties for all inner tags that are placed in the information card content area.

```
nav a span span{
  position: relative;
  font-size: 50%;
}
```

16. Content transition

It is the content areas of the information cards that will

need to change between showing their preview and full content when hovered, so we want to apply the ability to show transition animations. We use the same CSS reference and the transition property to set opacity and display properties and show transitions of one second.

```
nav a span span{
  -webkit-transition: opacity 1s, display 1s;
  transition: opacity 1s, display 1s;
}
```

17. Ensure preview visibility

The HTML is set so that the first inner element is used for preview content, hence you need to hide the second inner element until the information card is hovered over. This is done with :nth-child() CSS selector to hide the second inner element by default.

```
nav a span span:nth-child(2){
  opacity: 0;
}
```

18. Show main content

The information card needs to transition to show the main content it holds when it is hovered. This is just a case of using the :hover and the nth-child(2) selectors to target the content container.

```
nav a:hover span span:nth-child(2){
  opacity: 1;
  display: block;
}
```

19. Hide preview content

There is no need to display the preview content when the main content is visible, so we use the same strategy to reference the first element when the information card is being hovered over.

20. Highlight current card

Although the information card currently being hovered over expands, it still may be difficult for some people to distinguish the content of the current information card with its neighbours. This can be solved by making the background colour change - i we will make it white to ensure it stands out from the other information cards.

21. Image tidying

The examples in this tutorial have a blue background that looks out of place when the information card background changes colour to white. This can be resolved by adding a border to the right side of the image in the same shade of blue to fill in the space.

```
nav a img{border-right: 500px solid rgb(81,
173, 228);}
```

22. Cover the padding

The padding added to the information cards exposes the white background towards the left of the image icon. Fix this with a negative margin set to the size of the padding.

```
nav a img{
  margin-left: -1em;
}
```


Construct an adaptive printable CSS design

Make a self-adaptable content layout that is flexible and printable from the browser

tools | tech | trends HTML text editor such as Atom, and a HTML5 and CSS3 capable browser – tested on Safari, Chrome and Firefox



websites and web apps are considered a medium for providing on screen content, but are just as capable of delivering content ready for

printing. This alternative use for web-based content delivery opens new potential for the use of websites and

web-based applications. This tutorial takes a look at how web design principles can be applied to printing from the web browser through the use of adaptive design, graceful degradation and separation of design from content.

Although printable content is unlike on-screen content in that it is a static presentation format, we will take a look at how the delivered content can be supported

through JavaScript-based interactivity that enables users of a website or web-based application to select which version of the content to be printed – without the need for any duplication of the HTML-based content delivered.

The tutorial shows how a leaflet design can be modified using adaptive design techniques with CSS to become a voucher that is more ink efficient.

1. Get started

First, declare the main HTML page structure including the head and body section. The content container is an `<article>` element that has a `data-printVersion` attribute used later to define which print design is being used. Other elements will contain specific content sections.

```
<!DOCTYPE html>
<html>
<head>
<title>City Tours</title>
</head>
<body>
<article data-printVersion="">
</article>
</body>
</html>
```

2. Content overview

We are now ready to start inserting the main content. The leaflet will have a descriptive title and introduction in the `<header>` section of the HTML layout. This will be used to make it clear what is being promoted.

```
<header>
<h1>20% Discount</h1>
<p>Your luxury city tour awaits complete
with our 20% discount limited time offer.</p>
</header>
```

3. Action call

In addition to describing the promotion in the `<header>`

section, the offer needs to be presented in a simplified form that can be read at a glance to be printed. We add a new `<section>` element inside the `<article>` container with `class="outline"` that will enable it to be selected from the CSS for specific styling. Notice how the `class="screenOnly"` class can enable hiding from printing.

```
<section class="outline">
<ul>
<li>Amazing sights.</li>
<li>Incredible facts.</li>
<li>Free tourist guide.</li>

<li class="screenOnly">PRINT YOUR TICKET
NOW</li>
</ul>
</section>
```

4. Add informative content

The leaflet will present a section detailing highly informative information. This will not be part of the voucher version of the content, so a `.leafletVersion` class is applied to the `<section>` container for later detection in CSS. This section will also have some images defined via CSS using the attribute `data-image`.

```
<section class="leafletVersion content">
<div class="columns2">
<p>Lorem ipsum dolor sit amet...</p>
<p class="imageRow">
<span data-image="one"></span>
<span data-image="two"></span>
<span data-image="three"></span>
```

```
</p>
</div>
</section>
```

5. Terms and conditions

The leaflet must give clear details of the terms in which it can be used. A `<footer>` section is added to contain this content and will be styled with CSS. It is important that it is clearly visible to avoid any conflict between promotion and customers who use the voucher.

```
<footer>
<p>Valid on 01/02/03 before 12pm. No
refunds available.</p>
</footer>
```

6. Print interface

The preview screen will provide options to select which version of the content to print. The user interface requires buttons for each option, hence the HTML for these options will be placed after the closing `</article>` tag. These buttons activate the `printout()` function that then activates the page printing.

```
<div class="screenUI screenOnly">
Print options:
<input type="button" value="Print Voucher"
onClick="printout('voucher');"/>
<input type="button" value="Print Article"
onClick="printout('leaflet');"/></div>
```

7. Print buttons

The `printout()` function called by the previously



<Top left>

- The template now has the HTML elements and content in place, but lacks styling

<Top right>

- Printing option buttons created after styling is later added to make the button container float above page content

<Bottom left>

- Adding a background image adds style, but makes the text content more difficult to read

<Bottom right>

- Making the text white and adding a shadow helps the content stand out

inserted UI buttons needs to be defined in JavaScript in the <head> section. This function takes the version of the document to print as a parameter and applies it to the <article> tag's data-printVersion attribute so that CSS can detect the layout rules required for printing.

```
<script>
//.. Allow jQuery style DOM selection
$ = function(cssRule){
return document.querySelector(cssRule);
}
//.. Activate Page printing
printout = function(version){
$("[data-printVersion]").setAttribute("data-printVersion",version)
window.print();
}
</script>
```

8. Create resource files

The main HTML template content is now complete, but the additional CSS stylesheet resource files are still required to add the styling. Create a text file called 'styles.css' and another called 'styles_print.css', making sure that your text editor does not add TXT as a file extension. These also need attaching from the <head> section of HTML document.

```
<link rel="stylesheet" type="text/css"
```

```
href="styles.css" />
<link rel="stylesheet" type="text/css"
media="print" href="styles_print.css" />
```

9. Content container

Open the styles.css file you have created in your code editor. This file will be used to define styles that are common to both the on-screen preview as well as the print version. The <article> container is used to define the flow of the content to cover the full-page width, but adjust to however much content it contains, hence we don't need to define a height.

```
article{
display: block;
width: 90%;
clear: both;
margin: 1em auto 1em auto;
padding: 2em 5% 0 5%;
text-align: center;
}
```

10. Font sizes

A default font size is required for the <article> container, which can then be used by other elements to base their font sizes. Not only does this enable font sizes to automatically adapt themselves when the screen version is altered, but also when altering the font sizes for print when using a different type of measurement.

Hide unsuitable elements

Stylesheets can be used to control how different versions of the content are presented without the need for JavaScript or content duplication.

```
article{
font-size: 20px;
}
header h1{
font-size: 2.5em;
}
section.outline{
font-size: 1.8em;
color: #5AD8E5;
}
footer{
font-size: 0.5em;
}
```

11. Background style

Setting a background image as part of the on-screen preview will make the design look more appealing. We will want the content to stand out from the image, so using a semitransparent image, cover the full background, and set the background size as 'cover'.

HTML & CSS



<Top left>

- Text content is made to flow in two columns - this is suitable for static print content where users don't have a screen to scroll

<Top right>

- Adaptive images are now visible and can change to suit different print layouts if required

<Bottom left>

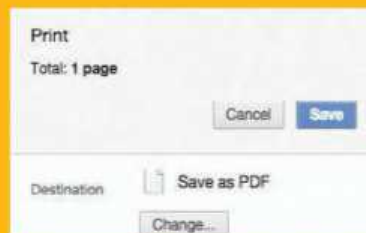
- The completed full page print preview of the leaflet version of the page content

<Bottom middle>

- The completed voucher version of the print layout shows a different design and eliminated content from the full leaflet

<Bottom right>

- Users must manually set the print margins from the browser print settings before printing if they want the content to cover the full page



```
article{background-image: url('img/
background5.jpg'); background-size: cover;
}
```

12. Adjust readability

The text may be difficult to read in some places with the new background. We can improve this by making the default text colour of `<article>` white to minimise the conflict between the colours used in the background, whilst each content area can make use of a dark transparent background to make the text stand out.

```
article{
text-shadow: 2px 2px #000;
color: #fff;
}
header, footer, section{
background: rgba(0,0,0,0.5);
}
```

13. Adding the padding

The final part that we want to add to the default CSS is the spacing to the different areas of text. Although this will increase the size of the voucher, it will also help people to distinguish the different parts of the content - making it easier for them to read and more likely that they will take notice.

```
header, footer, section{
padding: 0.1em;
}
```

14. Informative content layout

The section of the leaflet containing the informative content needs styling for its elements. With print content being static, text layout is often arranged in columns to enable easier reading. We will use CSS to provide this formatting with browsers that support it - those that don't can gracefully degrade the presentation to a single column of text.

```
.columns2{
text-align: left;
-webkit-column-count: 2; /* Chrome, Safari,
Opera */
-moz-column-count: 2; /* Firefox */
column-count: 2;
}
.screenUI{
display: block;
position: fixed;
top: 0;
left: 0;
height: 2em;
width: 100%;
}
```

```
color: #fff;
background: #777;
font-weight: heavy;
}
```

15. Informative images styles

The informative content area also has some CSS defined images that are placed in a container and these will use the `imageRow` class. The layout will make use of the `imageRow` container to keep the images separate from the two column text, with each `data-image` being a set size and displayed as an inline-block for horizontal positioning next to each other.

```
.imageRow{
display: block;
clear: both;
text-align: center;
margin-top: 3em;
}
.imageRow [data-image]:first-child{
margin-left: 0;
}
[data-image]{
display: inline-block;
width: 6em;
height: 6em;
}
```



```
margin-left: 3em;
background-size: cover;
}
```

16. Informative images styles

With the rules defined for data-image items, it is now easy to define the CSS rules for each individual image. This will be achieved using the value of the data-image items.

The advantage of defining images in this way over the traditional tag is that we can use CSS to change the images for different versions of the print design if required.

```
[data-image="one"]{
background-image: url('img/photo1.jpg');
}
[data-image="two"]{
background-image: url('img/photo4.jpg');
}
[data-image="three"]{
background-image: url('img/photo5.jpg');
}
```

17. Initiate print styles

We are now ready to add the parts of the CSS that optimise the page layout for the different print options. Open the styles_print.css file you created and define the CSS rules that will hide elements not required for the selected version of the design - notice how 'important' is used to make sure these rules take priority over any other CSS rules.

```
.screenOnly{
display: none !important;
}
[data-printVersion="leaflet"]
.voucherVersion{
display: none !important;
}
[data-printVersion="voucher"]
.leafletVersion{display: none !important;
}
```

18. Size and margins

We want to make sure that there is no margin or padding getting in the way of print, whilst also making sure that the base font size used by the article container is using a measurement that is ideal for print - in this case setting the base font size to 14pt.

```
body{
margin: 0 !important;
padding: 0 !important;
}
article{
font-size: 14pt;
margin:0;
padding: 0 5% 0 5%;
}
```

19. Full page leaflet

The leaflet needs to cover a full page, hence we make use of the 'data-printVersion="leaflet"' attribute to define specific alterations to the standard presentation.

In this case, we want to set a height and padding of the <article> container to equal 100 per cent, with the inner .content class element being made to have a set height that covers the required space.

```
article[data-printVersion="leaflet"]{
width: 90%;
height: 95%;
padding-top: 5%;
}
[data-printVersion="leaflet"] .content{
text-align: left;
height: 55%;
}
```

20. Optimise presentation

The promotion image used for the screen preview version of the voucher will use unnecessary ink when printing, therefore a more printer-friendly presentation of the image should be used.

This requires the backgrounds and font shadow of <article>, <header>, <section> and <footer> containers to be reset. We'll also set the <article> container to have a border.

```
article[data-printVersion="voucher"]{
background: none;
border: 6pt solid #000;
color: #000;
text-shadow: none;
width: 85%;
margin-left: auto;
margin-right: auto;
}
[data-printVersion="voucher"] header,
[data-printVersion="voucher"] footer,
[data-printVersion="voucher"] section{
background: none;
}
```

21. Define voucher imagery

With the main background image now gone, we need to activate the replacement. The <section> elements can be used to define how their content are presented; class="outline" will be shown with background image, whilst the class="content" will be hidden from the voucher because it is labelled with a .leafletVersion class.

```
[data-printVersion="voucher"] section.
outline{
background-image: url('img/background5.jpg');
background-size: cover;
color: #fff;
text-shadow: none;
}
```

Code Library

JavaScript print page options

Let JavaScript use jQuery style CSS rules to select HTML page elements to perform operations on.

```
$ = function(cssRule){
return document.querySelector(cssRule);
}
printout = function(version){
$("[data-printVersion]").
setAttribute("data-
printVersion",version)
window.print();
}
[data-image]{
display: inline-block;
```

Adjust article container to have the data-printVersion attribute used by CSS to identify how to print the page.

```
width: 6em;
height: 6em;
margin-left: 3em;
background-size: cover;
}
[data-image="one"]{
background-image: url('img/photo1.jpg');
}
[data-image="two"]{
background-image: url('img/photo4.jpg');
}
[data-image="three"]{
background-image: url('img/photo5.jpg');
}
```

An alternative to tag enables CSS to define image elements where their source image and settings can be changed for different print layouts.

22. Prepare for printing

Unfortunately browsers don't provide access to changing print settings from JavaScript or CSS, meaning that users must be aware of making any required adjustments. This may be required for instances such as when you only want the background image printing or if you want to set the design to print without any page border or padding.

Users could be provided a notice about this when printing through an alert triggered placed in the first line of the printout() function.

```
alert("Make sure to alter your print
settings if you don't want page borders.");
```

jQuery & JavaScript

- 70 The new JavaScript standard
Get to know the new standards
- 76 jQuery special effects
Amplify your jQuery
- 82 Make dynamic graphics with the
p5.js library
Create interactive drawings
- 86 Create shuffling text effects with
jQuery
Add interesting text effects
- 88 Design a 2D game using the Pixi
engine
Render a simple game in WebGL
- 92 Sync animations to audio and
video with Popcorn.js
Trigger events easily
- 96 Animate SVGs with the Vivus.js
library
Create high-quality line drawings
- 100 Code on-scroll image animations
Emphasise parts of your page
- 102 Create animated infographics
with Snap.svg
Add interactivity
- 106 Prototype apps with
Framer.js and
Framer Studio
Add animations

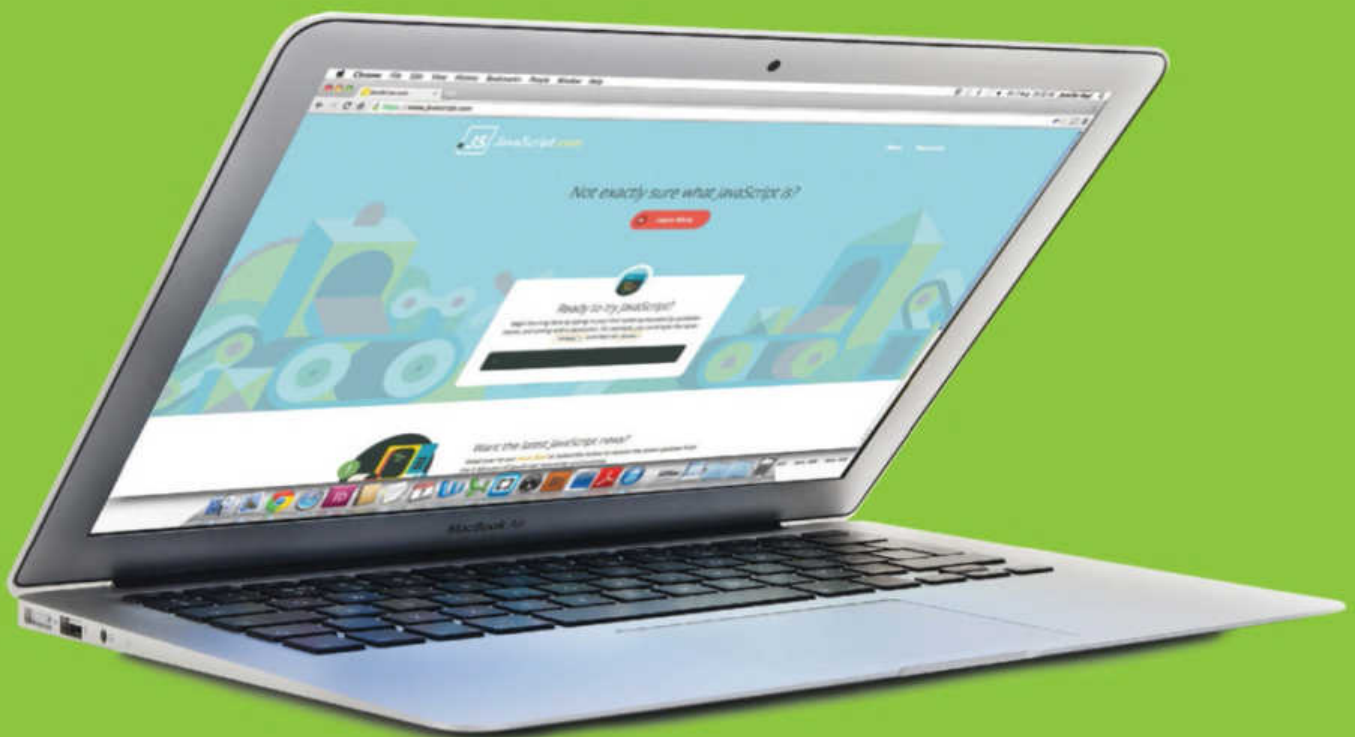




↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

THE NEW JAVASCRIPT STANDARD

Upgrade your applications with ECMAScript 6 and
boost your scripts with the newest features



ES6 AND HOW IT IMPACTS JAVASCRIPT

JavaScript is available in a variety of forms today like that of web browsers, native applications, server-side scripting and even more exotic environments. As a language, it is steered by a standard called ECMAScript, which sets the core syntax and features of JavaScript. As each version of the ECMAScript standard is ratified, JavaScript engines move to support the changes detailed within it. This provides JavaScript with a general compatibility between implementations of the language. Code written to work in web browsers, that aren't tied to anything browser specific (like the DOM), should work in other JavaScript engines like Node.js.

ECMAScript 6 (referred to as ES6) is the next version of this standard for JavaScript and is nearing its final release, meaning that the features are set and that implementations are fully detailed and unlikely to change in any significant way. Browsers, scripting engines and other applications have already started adding support for ES6 with Firefox and Chrome leading the way. Once it's been officially finalised, expect full support of the standard to be taken up quickly by the major applications. ES6 will introduce new syntax and features to JavaScript, designed to make life easier for developers and solve common programming problems. The goals state that it intends to make the language better for writing complex applications, libraries and code generators so they've included a module system that is reusable and can be used to improve code organisation. Class support is now built in with its own syntax, and methods have been added to help manage asynchronous code, such as promises and generators.

At lower levels there are new variable types to provide alternatives to objects and arrays for your data, known as 'sets' and 'maps'. Building strings from several variables has also been improved through a new template string syntax, which can also be used to easily create multiline strings without being split. Defining functions has also received attention through a new 'fat arrow' syntax, which can help with the often confused nature of the 'this' scope in relation to the function being called. Recursive functions can take advantage of performance improvements by returning at their end. Block-scoped variables, constants, default parameters and much more make this quite the upgrade for JavaScript.

What does this all mean for developers? On offer is better code organisation, plenty of new shortcuts for typical tasks, standard formats for sharing libraries and much more. There's also nothing holding you back from using the major features of ES6 now. Applications are available that will take your ES6-based code and compile it into ES5, making your code compatible with a wide range of JavaScript engines that are in use today.



Matt Gifford – consultant developer, Monkeh Works Ltd (monkehworks.com)

“ES6 is going to empower JavaScript developers, giving us the means and flexibility to write modular, well-organised code that is easily scalable, testable and understandable. Well-structured code will improve application architecture and will have a direct positive impact on performance.”

SUPPORT

Some ES6 features are commonplace, others are unsupported, but there are workarounds available

Browsers

Firefox and Chrome have made significant progress with support for these new features of JavaScript. They now support over 40 per cent of the new standard (at the time of writing). Especially in the area of new variable types, improve function syntax and promise support. Internet Explorer support is lower with IE11 only covering 11 per cent, but the next release will bring in wider coverage.

Scripting engines

The Node.js engine covers a quarter of the ES6 standard in its recent stable release. However, a recent fork of the project, named io.js, has pushed

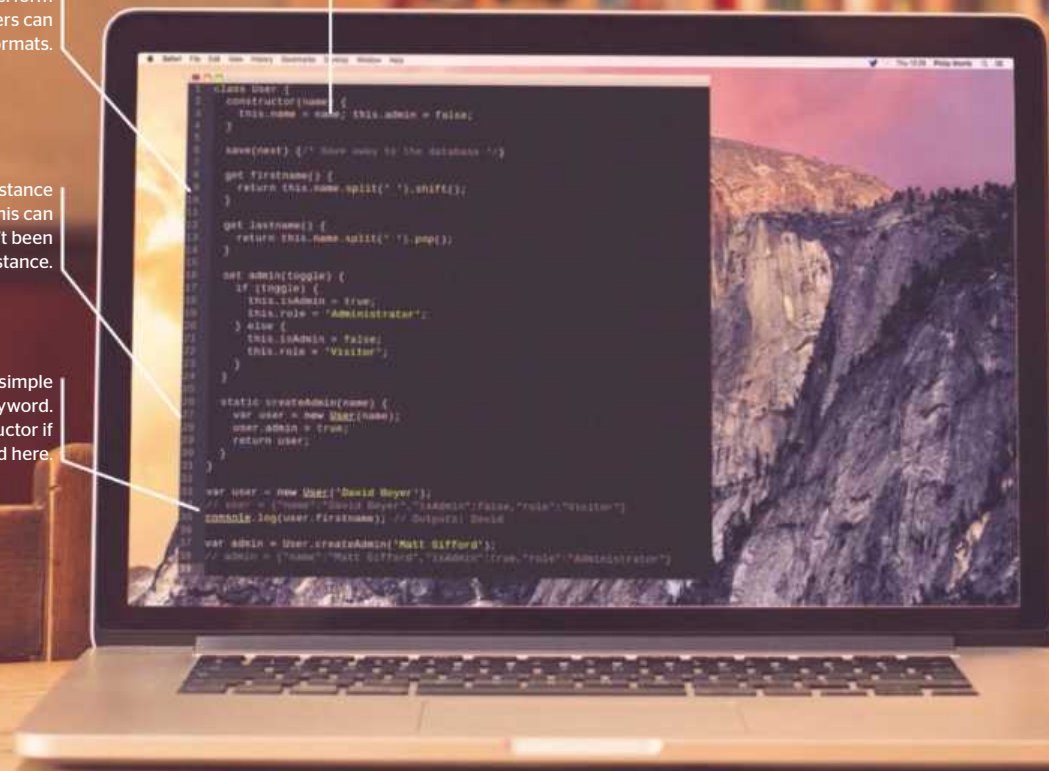
this to over 40 per cent. There are discussions going on to possibly bring those changes back into Node.js or the two projects to work together, so expect ES6 support to grow further within these engines.

Transpilers

Due to lack of ES6 support or missing features in certain JavaScript environments, applications have been created that take your ES6-based code and compile it into ES5-compatible code. Through polyfills, these transpilers use alternative ways to replicate ES6 behaviour. Your original source code will always be available as you wrote it, but you'll have a new copy to distribute to ES5 environments. Babel (babeljs.io), at the time of writing, can support 76 per cent of the ES6 standard within ES5 engines. As the ES7 standard is being formed, effort has gone into Babel to provide early support of some features.

A CLOSER LOOK AT THE CODE

1. The constructor is executed when a new instance is created from the class, providing the ability to set anything up that we might need within the instance.
2. These setters enable a tighter control of what data may be set and even perform actions based on it, while the getters can provide the instance data in several formats.
3. Static methods aren't present on an instance of the class but rather the class itself. This can be used for any unique logic that hasn't been tied to a single instance.
4. Creating a new instance of a class is simple enough through the use of the 'new' keyword. Data can also be provided to the constructor if it is required here.



THE MAJOR FEATURES OF VERSION 6 CLASSES

New to JavaScript, classes provide a shortcut to OO-type code organisation

With ES6, JavaScript will finally have classes. These aren't new in the programming world and it has also been possible to replicate classes in JavaScript already, but having them built into the language by default will encourage a standard way of using object-orientated code, making it easier for you to write large applications and also understand other people's code.

A class at its core is used to create a template for an object, which can contain its own data and methods. ES6 will support several concepts that work with classes to aid in organisation of your code and provide functionality. Inheritance is included, so you can base a class off another existing one and make use of it, within your new class via the 'super' variable. Instance

methods are standard, and they're available when you create an object from the class template, but static methods can also be created and used on the class template itself. A constructor method is provided, enabling you to perform tasks on an instance of your class template, whenever one is created. Finally, there are getters and setters and these will provide the ability to capture, with a class method, when data is set or retrieved against your instance.

New function syntax

The 'this' scope can sometimes cause confusion within JavaScript. With the established function syntax, the 'this' scope can be different depending on where the function is being executed. If you wanted to retain a function's 'this' scope, you would have to bind a value to it with '.bind(this)'. The new syntax will create a function bound to where it was created, with the original 'function' method remaining available.

```
this.sides = 6;
var rollNumbers = (dice) => {
  var rolls = [];
  for (let i = 0; i < dice; i++) {
    rolls.push(Math.floor(Math.random() * (max + 1 - min)) + min);
  }
  return rolls;
};
```

Template Strings

Normally, to construct a string in JavaScript based upon set text and variable values, you either concatenate them using the plus operator (+) or join them together through an array. ES6 provides a new syntax known as template strings. These will enable you to create a string with variables inside it, avoiding the need of having to stitch it together. They also provide some other useful features, such as being able to create

“With the established function syntax, the ‘this’ scope can be different depending on where the function is being executed”

Let and const

Declaring variables is now more flexible. Constants are now available by prefixing with 'const', enabling you to declare and set a constant value to an identifier. If any attempt to change the value is made, an error will occur, safeguarding the data. You can now also create a variable that will only exist within the block, statement or expression it is declared within through use of the 'let' keyword instead of the normal 'var'. This helps with creating safer, self-contained code and avoids overwriting variables used within the same function. With considered use this could also result in lower memory usage, as 'let' variables could be freed sooner.

```
const apiKey = process.env.API_KEY;
apiKey = 'something'; // This would throw
an error
for (let i = 0; i < 10; i++) {console.
log(i);}
// The i variable only exists within the
"for" loop
```

Modules

One major missing feature from previous versions of JavaScript has been a built-in module system. As ES6 intends to provide a better language for building complex applications and libraries, it makes a module system an important piece of the puzzle. The JavaScript community has created its own alternatives such as Asynchronous Module Definition (AMD) and CommonJS (CJS) but with ES6, JavaScript will finally have one built in.

With the module system, anything within a file won't be available outside of it, unless you export it. Through use of the 'export' keyword, you can specify variables (data, functions, classes) that you wish to make use of elsewhere. Then, from another file, you can import some or all of those exposed items. This forms the basis of organising your application code, splitting it apart and making it more reuseable.

```
// mag.js
const title = 'Web Designer';
export default function() {
return {
title: title,
data: getMagData()
};
};
// index.js
import Mag from 'mag';
var data = Mag();
```

Function parameter additions

Functions having parameters is nothing new within JavaScript, but ES6 has added support for default values and gathering some arguments as an array. A function in JavaScript can have numerous parameters available, but when calling the function you aren't required to supply every single one. Those arguments

MAKING A PROMISE AJAX REQUEST

An example of how to use promises to wrap asynchronous requests

In this example promises are primarily used from ES6, with a few other minor features here and there. Targeting the browser with our code, we have XMLHttpRequest available for making AJAX requests for further data. In this example we need to pass an application object a list of countries and languages that are available in JSON-based files.

The files could be requested one at a time, but this would not be as fast as requesting both at once (asynchronously). This would take some effort to manage normally, but with ES6 promises, it becomes a lot simpler. The first and most important step is to wrap a call to the XMLHttpRequest object with a promise. When a new promise is created, it provides two functions to define whether the promise was successful or failed via 'resolve' and 'reject'. Anything passed to 'reject' will cause the promise to fail and fire any function provided to the .catch(fn) method (of the returned promise). If the request worked, we'll attempt to parse the JSON and fulfil the promise with 'resolve', passing it the parsed data object.

Now, any call to our XMLHttpRequest object wrapper (fetch), will return a new promise for the requested URL. This promise will remain unfulfilled (pending) until it either shows an error (rejected) or alternatively, it shows that it has received the response and has parsed correctly. Using the returned promises' .then method with a function will gain access to the parsed data.

To easily make multiple requests at the same time and wait for all of them to return, a further promise can be created. Promise.all will take an array of other promises (and values) and wait for all of them to resolve before it fires anything attached to .then. It can also manage errors across all of those multiple promises.

EXPERT COMMENT



BRIAN LEROUX

Developer at Adobe
bit.ly/1wadOsK
@brianleroux

ECMAScript 6 is the JavaScript we've been waiting for. It reveals the hidden beauty of JS, taking cues from contemporary dynamic languages and adding its own shortcuts paved from common paths tread by the developer community. The best part - we can start using ES6 today in io.js, and with the help of Babel, anywhere that runs ES5 (which is just about everywhere).

a string over multiple lines in your source code, something that isn't normally possible without splitting your string apart and joining it with the plus operator at the end of each line.

```
var luckyNum = Math.round(Math.random() *
100);
var output = 'Today's lucky number will be
${luckyNum}.';
```



jQuery & JavaScript

that aren't supplied would normally remain 'undefined', but now there's the ability to set default values for them. ES6 also enables declaring of some function parameters and then having the rest grouped as an array. This is useful if you want to let your function take a flexible amount of arguments, but have a few initial set ones.

```
function plural(word, amount, suffix='s') {
  return (amount === 0) ? word :
    `${word}${suffix}`;
};
plural('test', 5); // returns 'tests'
function setNames(userId, ...names) {
  // names would equal ['Bob', 'Robert',
  'Bobby']
};
setNames(42, 'Bob', 'Robert', 'Bobby');
```

Promises

Sometimes, in a JavaScript environment, you'll have to do something asynchronously. This could be in a web browser where you need to make an AJAX request, or in Node.js when making a request to a database, or handling web requests. The standard way to handle such tasks would be to provide a callback function, which would be executed when the task was complete (like data being received from the AJAX request).

Promises are an alternative approach to handling this asynchronous tasks, which can make code much more readable. In comparison, callbacks can sometimes lead to pyramid-type code where callbacks are nested by several levels. Promises work as you might expect. You create a task (usually asynchronous in nature) which returns an unfulfilled promise. When the task completes, the promise is fulfilled and it'll execute anything attached to its 'then()' method. If a task fails with an error, the 'catch()' method will be fired. Promises can be easily passed around, enabling you to check its state and require one promise before others are completed.

```
'g'
function initApp() { `${word}${suffix}`;
return new promise(function(resolve, reject)
{
  // App initialization...
  if (app.complete) return resolve(app.id);
  reject(new Error('App failed to
  complete')); });
}
initApp()
.then(function(appId) {
  // Now setup the UI
})
.catch(function(error) {
  // Handle the error
  // Handle the error
});
```

FROM ES6 TO ES5

Take your ES6 code into ES5 environments

Babel is a JavaScript-based application to take your ES6 code and make it compatible with ES5 JavaScript engines, that are missing some or all of the features introduced by ES6. Grunt can be used to manage this task and hold any configuration it may need. In the screenshot below, the steps are shown for creating a project and then the Gruntfile for executing Babel against the ES6 code.

Once the Gruntfile is ready, ES6-based source code can start to be created within the src directory. When ready, a single Grunt command will use Babel to build the ES5 code. Babel attempts to keep the resulting code as clean as possible, but it will be using workarounds for those missing ES6 features. This includes using polyfills to add support and providing source maps for easier debugging.

The screenshot shows the Babel website with a yellow header containing the Babel logo and navigation links: Learn ES2015, Setup, Usage, Advanced, Try it out, FAQ, Community, Blog, Twitter, and GitHub. The main heading is "Babel transforms your JavaScript". Below this, it says "You put JavaScript in" followed by a code snippet: `myOldWeirdJavaScript("foobar");` and "And get JavaScript out" followed by: `myNewTransformedJavaScript("yay!");`. A section titled "ES2015 and beyond" states: "By default, Babel ships with a set of ES2015 syntax transformers. These allow you to use new syntax, right now without waiting for browser support. [Learn more](#) →". To the right of this text is a list of supported features: Array comprehensions, Arrow functions, Async functions, Async generator functions, Classes, Class properties, Computed property names, Constants, Decorators, Default parameters, Destructuring, Exponentiation operator, For-of, Function bind, Generators, Generator comprehensions, Let/const, Modules, Module export extensions, Object rest/spread, Property method assignment, Property name shorthand, Rest parameters, React, Spread, Template strings, Type annotations, and Unicode regex. Below this is a section for "JSX and React" stating: "Babel ships with built-in support for JSX. Use it together with the [babel-sublime](#) package to bring syntax highlighting to a whole new level." At the bottom of the page is a dark navigation bar with links: Promises, API Reference, Patterns, Generators, Implementing, and Specification.

Promises

by Forbes Lindesay

Motivation

Consider the following synchronous JavaScript function to read a file and parse it as JSON. It is simple and easy to read, but you wouldn't want to use it in most applications as it is blocking. This means that while you are reading this file from disk (a slow operation) nothing else can happen.

```
function readJSONSync(filename) {
  return JSON.parse(fs.readFileSync(filename, 'utf8'));
}
```

To make our application performant and responsive, we need to make all the operations that involve IO be asynchronous. The simplest way to do this would be to use a callback. However, a naive implementation will probably go wrong:

```
function readJSON(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) return callback(err);
    callback(null, JSON.parse(res));
  });
}
```

- The extra `callback` parameter confuses our idea of what is input and what is the return value.
- It doesn't work at all with control flow primitives.
- It doesn't handle errors thrown by `JSON.parse`.

We need to handle errors thrown by `JSON.parse`, but we also need to be careful not to handle errors thrown by the `callback` function. By the time we've done all of this our code is a mess of error handling:

EXPERT COMMENT

**JACK FRANKLIN**

Engineer, GoCardless
jackfranklin.co.uk
@Jack_Franklin

I'm personally really excited about ES6, not only for the features it brings but because it's getting such backing from developers already. We're seeing tools spring up to let us experiment and use these new features today in production applications, without having to wait for browser support to catch up.

“Promises are an alternative approach to handling this asynchronous tasks”

```

1 //
2 npm init
3 npm install --save-dev grunt
4 npm install --save-dev grunt-hbabel
5 npm install --save-dev grunt-contrib-cssmin
6
7 Create .gruntfile.js with the following.
8
9 //js = ES6 source code
10 //babel = ES5 converted code with source maps.
11
12
13 module.exports = function(grunt) {
14   grunt.initConfig({
15     "clean": ["build"],
16     "babel": {
17       options: {
18         sourceMap: "inline",
19         optional: ["runtime"]
20       },
21       build: {
22         files: [{
23           expand: true,
24           cwd: "src",
25           src: ["**/*.js"],
26           dest: "build"
27         }]
28       }
29     }
30   });
31   grunt.loadNpmTasks('grunt-babel');
32   grunt.loadNpmTasks('grunt-contrib-cssmin');
33   grunt.registerTask('default', ['clean', 'babel']);
34 }
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```




DOWNLOAD TUTORIAL FILES

www.filesilo.co.uk/bks-747



JQUERY

SPECIAL EFFECTS

Amplify your jQuery and elevate your websites.
Discover the best plugins for giving your project
next-level visuals



GET GOING WITH NEON

Getting started with NoVacancy is really easy
github.com/chuckyglitch/novacancy.js

Neon can lighten up boring spaces. Their signs adorn cheesy shops and upscaleness can be proportional to the number of burnt-out bulbs, so let's get some!

INCLUDE THE LIBRARIES

Being a jQuery plugin, NoVacancy requires its mother library along with a copy of the actual source code for the neon blinking. NoVacancy is currently hosted at GitHub, so copy it to your web server.

```
<script type="text/javascript" src="http://
ajax.googleapis.com/ajax/libs/jquery/1.9.1/
jquery.min.js"></script>
```

```
<script type="text/javascript" src="jquery.
novacancy.js"></script>
```

INSERT ELEMENT

NoVacancy animates one or more elements created from `` tags located inside of a `<h1>` headline. You can set font styles and background colours in an accompanying CSS stylesheet, try Google's Monoton.

```
<h1>
<span id="1A">AAAAA</span>
<span id="1B">BBBBB</span>
</h1>
```

INVOKE METHOD

Starting NoVacancy can be done by invoking the

`novacancy()` method on a reference returned from jQuery's `$()` function. Blinking can be controlled via a trigger that lets you turn blinking on and off.

```
$('#no').novacancy();
$('#no').trigger('blinkOn');
$('#no').trigger('blinkOff');
```

SET PARAMETERS

Novacancy() can take a JSON object with settings, and thereby let you adjust the behaviour of the plugin. For example, the blinking frequency can be decreased to give the site a 'calmer' appearance.

```
$('#no').novacancy({
  'reblinkProbability': 0.1,
  'blinkMin': 0.2,
  'blinkMax': 0.6,
  'loopMin': 8,
  'loopMax': 10,
  'color': 'ffffff',
  'glow': ['0 0 80px #ffffff', '0 0 30px
000', '0 0 6px #0000ff'],
  'off': 1,
  'blink': 1,
  'classOn': 'on',
  'classOff': 'of',
  'autoOn': true
});
```

EXPLODE

bit.ly/1MgJzHP

Explosion effects provide a simple and effective way to draw the user's attention to any facts. This plugin provides a ready-made effect composed of a group of red tiles. Invoking it is as easy as calling the `explosion()` function on the element which will act as 'host':

```
$("#body").explosion({
  origin:{
    x:xClick,
    y:yClick
  },
  particleClass:"particle"
});
```

FALLING SNOW

workshop.rs/2012/01/jquery-snow-falling-plugin

Snow is fun and not only during the winter.

Adding snowflakes can be accomplished by generating a group of particles, which are then moved across the screen slowly. Alternatively, use this plugin – just one call changes the weather:

```
$.fn.snow({ minSize: 5, maxSize: 50,
newOn: 1000, flakeColor: '#0099FF' });
```

POP-UP WINDOWS

inserthtml.com/2013/08/modal-image-slider

Image galleries can be simple but adding in pop-up windows can also add something extra.

This is especially true if you use them in an effective way to present large amounts of data in a compact fashion. This plugin can handle both aspects of the problem: it takes care of bringing the little images on screen, and furthermore will provide you with an attractive animated pop-up window with detailed information.

SCROLLER COASTER

joelb.me/scrollpath

Traditional websites are scrolled in a linear fashion. This plugin permits you to change the scroll flow: when scrolling, your users will find content flowing left to right, up and down and even in a circular fashion. It is no longer maintained, but works well on all widely supported browsers.

BURN EFFECT

sinetheta.github.io/burn

This plugin adds a flame-like effect to text on your website, making use of the human age-old conditioning related to burning objects. Its flexible physics engine can be configured to create blue, green or yellow fires of varying intensity.



TEXT ANIMATION

plugins.jquery.com/letter-drop

Users hate websites that are slow to load. Adding animations is a sure-fire way to sweeten the wait. This plugin permits you to deploy a cool letter-drop effect and your headlines will appear to fly in from the top. Invoking the letter drop is as easy as calling the `letterDrop()` function on an element of choice. It should be provided with a set of CSS parameters.



VINTAGE TEXT TYPING

github.com/ericterpstra/jqVintageTxt

Traditional terminals are a simple and convenient way to display command-line systems. This plugin provides a ready-made virtual terminal, which comes complete with its own input handling. While its practical usage might be limited, hipsters will not be able to resist its retro awesomeness.

COLOUR CHANGE

henrygd.me/colorflow

Smart bulbs are popular due to their ability to change the current light colour in a smooth and perfectly animated fashion. This plugin provides the maths, making gradation-style colour changes a single method invocation.

SCROLLER

bit.ly/16YtLZx

Scrolling LED displays give websites and web apps a steampunk-esque look. This plugin takes care of everything: you pass in a text and a target element and the rest is handled for you. Large panels take up quite a bit of processing power – this is not suited for performance-sensitive pages.

PARALLAX SHADOWS

jimmylocoding.com/lilt

Static shadows are so 2009. This plugin creates shadows that changes their position as the user scrolls across the page. Implement it by adding the `lilt` class to elements deemed worthy of any 'shadowy attention'.

3D FOLD SCROLL

github.com/nickavignone/responsive_3dfoldscroll

Nick Avignone's plugin divides your website's content into a group of 'panes'. Scrolling is then accomplished by collapsing and unfolding panes in a perspective-correct fashion. Using the product is not recommended on a website that contains large blocks of text which must be read continuously. This is because the scrolling and folding effect means that any important information is folded away as you read.



AIRPORT FLIPBOARD

github.com/zemax/jquery-splitFlap

One of the most iconic images of the since-overhauled Baghdad Airport was its split-flap flight information system. This plugin permits your websites to provide a similar effect with options to set

the flaps to scroll upon a mouse click. Be warned though, the performance demands of this plugin are extreme and it will struggle on lower-powered desktop machines and devices.



github.com/maroslaw/rainyday.js

Animated backgrounds might be distracting, but artistically inclined readers will love them nevertheless. RainyDay is an impressive plugin which simulates rainfall on your screen, but be wary as it's resource hungry.

LET IT RAIN!

Adding a svelte rain effect can be accomplished in six easy steps

FIND A BACKGROUND

RainyDay works best when combined with night shots showing a busy city with well-lit buildings. If you don't have such an image at hand, use Flickr's creative commons search to look for one. But do keep in mind that attributions must be given under all circumstances to avoid copyright issues.

INSERT TAGS

Adding the library is to be accomplished via a `<script>` tag. Its magic will be done in a `` tag, which should be pointed to the image obtained in the step before.

By default, RainyDay will apply a blurring effect automatically to your image.

```
<script src="dist/rainyday.0.1.1.min.js"></script>

```

CREATE ENGINE

RainyDay is managed via an instance of the engine class. It must be provided with a JSON array containing at least the image attribute pointing to the `` element. You can pass in further attributes to modify rendering behaviour.

```
var engine = new RainyDay({
  image: element,
  parentElement: someDiv,
  crop: [0, 0, 50, 60],
  blur: 10,
  opacity: 1
});
```

ADD RAINDROPS

Individual raindrops are spawned by the `rain()` method. It takes an array of tuples describing drop size and drop count, and furthermore accepts a numerical parameter describing the amount of milliseconds that must elapse between each spawning.

```
engine.rain(
  [
    [1, 0, 20],
    [3, 3, 1]
  ],
```

```
);
});
```

LEAVE A TRAIL

Raindrops don't flow down windows without leaving a trace. RainyDay provides a group of ready-made constants that permit you to change the way your drops move. Be aware that longer trails increase CPU load significantly.

```
engine.trail = engine.TRAIL_SMUDGE;
```

MIND OVER MATTER

By default, RainyDay will use linear gravity pointing downwards. You can modify the direction of the gravity vector in the engines initialisation. Finally, the type of gravity can be modified by setting the engine's gravity attribute to `GRAVITY_NON_LINEAR` or `GRAVITY_NONE`.

```
var engine = new RainyDay({
  image: this,
  gravityAngle: Math.PI / 9
});
```

It provides ready-made constants

IMAGE ZOOM

github.com/waynegr/imgViewer

One well-placed image says more than a thousand words. However, presenting figures always requires compromise: large images take up a lot of valuable screen real estate.

This innovative plugin provides users with a scroll viewer similar to the one that is used on desktop operating systems. Drag and drop is used for modifying the visible area,

while the mouse wheel permits users to zoom in and out flexibly.

It is worth remembering that not all users will have access to a mouse with a scroll wheel. A prime example would be users working on laptops. So when implementing this plugin, be mindful of who will be using the site and if they are able to take advantage of the zoom effect the plugin offers.



MULTIPLE SCREENS

ian-devries.com/multiscreenjs

Multiscreen workstations permit users to keep multiple things in view at the same time. This plugin tries to mimic this by creating multiple virtual pages. You must then provide links that permit the user to switch between the individual views.

Each view is composed of a <div> tag, which must be provided with a class attribute of ms-default in order to be 'attractive' to the plugin code:

```
<body>
<div id="screen1" class="ms-
container ms-default">
```

```
<!-- screen1 content -->
</div>
<div id="screen2" class="ms-
container">
<!-- screen2 content -->
</div>
</body>
```

Designers will cleverly combine Multiscreen with a hot key-based navigation so that websites displaying stock quotes or similar information can use this process to present their users with multiple views that are independent from one another.

FLIP BOOK EFFECT

turnjs.com

Whether you like it or not, eBooks are hot. Provide a stack of images which show the individual pages of a printed document, and this nifty plugin will take care of the rest.

The book view enables users to scroll through the individual pages with realistic page-flip animations. Putting

the mouse pointer into page corners reveals a smoothly animated handle and you can scroll through the content with arrow keys.

Since the website is made up of individual images, any users that don't know about wget will have a hard time saving (and pirating) your content.



ANIMATED TEXT MASK

jqueryscript.net/text/Animated-Text-Mask-Background-with-jQuery-CSS3.html

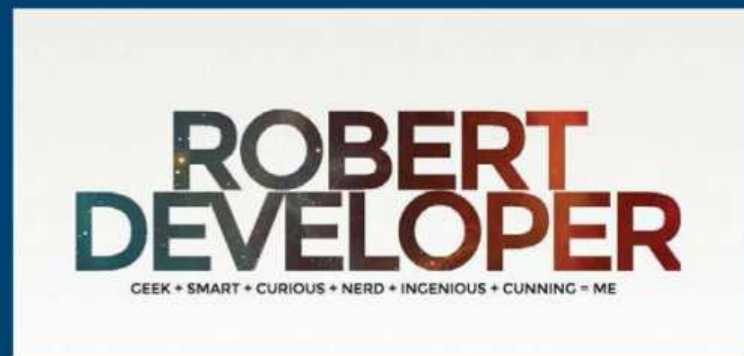
Some effects are so wacky that they can barely be described in words. This plugin takes the outline of a font, and then 'fills' it with a permanent varying part of a background image.

Please be aware though that the motion logic is not contained within this plugin. Developers must provide it in a method such as the one that we have written below:

```
$(document).ready(function(){
var mouseX, mouseY;
var ww = $( window ).width();
var wh = $( window
).height();
var traX, traY;
$(document).
```

```
mousemove(function(e){
mouseX = e.pageX;
mouseY = e.pageY;
traX = ((4 * mouseX) / 570) +
40;
traY = ((4 * mouseY) / 570) +
50;
$(".title").css({"background-
position": traX + "%" + traY +
"%"});
});
});
```

Note that Animated Text Mask does not currently support all the major browsers. Chrome and Safari will present the plugin's full potential whereas Firefox and Internet Explorer do not currently.



ERGONOMICS ARE IMPORTANT

We've presented a selection of impressive plugins which permit you to create websites with movie-worthy effects. Please keep in mind that overdoing animation is harmful: users with slow PCs will find their browsers locked, while sufficiently fast PCs will have your readers distracted from the content at hand.

ANIMATED COUNTDOWN CLOCK

class.pm/files/jquery/classycountdown

In today's fast-moving industrial environment, rumours and prerelease announcements can make or break a company. This toolkit ups the ante by providing a set of fancy-looking countdown timers which can really drum up attention.

ClassyCountdown does its magic by abusing a dial component. Thus, you must include the knob plugins along with the CSS stylesheet. The actual widget is then created like this:

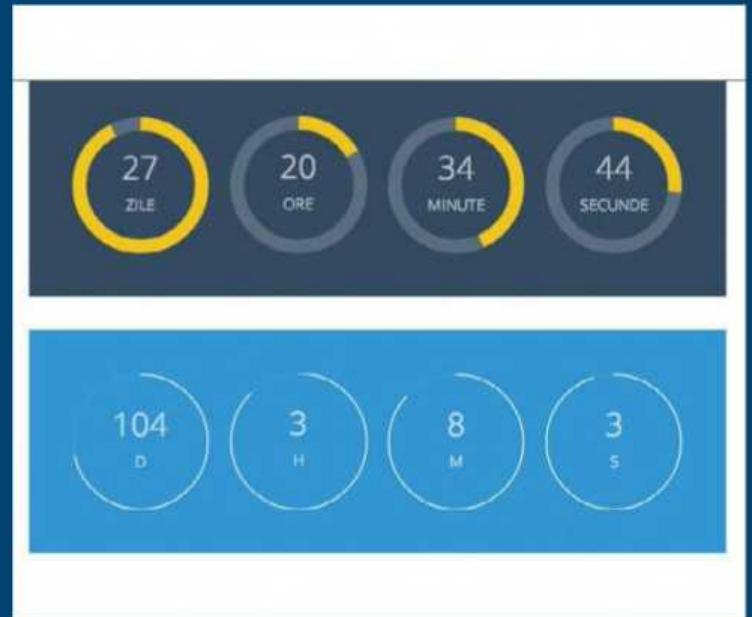
```
<div class="countdown"></div>
```

Starting the actual countdown process can be accomplished by invoking the

```
ClassyCountdown method:ww
$('.countdown').
ClassyCountdown({
  theme: "flat-colors",
  end: $.now() + 10000
});
```

Initialising a ClassyCountdown widget requires you to pass in two parameters. 'Theme' informs the rendering engine about the colour scheme, whereas 'end' is responsible for setting the end time.

Please keep in mind that the timer does its magic using the system time. But be aware that malicious users can manipulate it with ease in order to reveal any of the hidden information.



MULTIPLE TEXT EFFECTS

jschr.github.io/textillate

Motion makes otherwise-bland texts more interesting. This plugin provides you with a set of canned animations which can be applied to the blocks of prose littered across your web site.

Adding Textillate requires you to include a total of three different libraries. One or more items must be provided with the 'tlt' class in order to mark them as 'willing to be moved':

```
<h1 class="tl" >My Title</h1>
```

Starting the actual animations is then accomplished by invoking textillateO:

```
$(function () {
  $(' .tlt').textillate();
});
```

Further customisation can be done via tags. Alternatively, textillateO can also take a JSON array with settings:

```
<h1 class="tl" data-in-
effect="rollIn">Title</h1>
```

Motion on the page can be distracting. It is important to think of the user experience when implementing any type of animation.

WATER RIPPLES

alligatr.co.uk/lake.js

Be it Lake Balaton, the coast off Portorož or the thermal sea of Hévíz: water is tranquil and can calm people down. Lake.js takes a random image, and expands it with a nicely animated water effect.

The lake.js plugin can be deployed on top of any tag. Its first act will involve the creation of a canvas tag on top of the image, and this will then be used to display the actual waves that we want displayed.

```
$('#lake-img').lake({
  speed': 1,
```

```
'scale': 0.5,
'waves': 10
});
```

Three parameters will let you change the speed and the intensity of the waves. The default values shown above work well - change them to suit you.

Lake.js works with precomputed frames. This means that the product will take some time after loading with the effect, but afterwards it will then play back the actual animation with amazing speed.

ORIDOMI

oridomi.com

Ever since the Chinese developed paper, folding it has been an endless source of pleasure. OriDomi lets you create parts of your site to fold and unfold in response to external events.

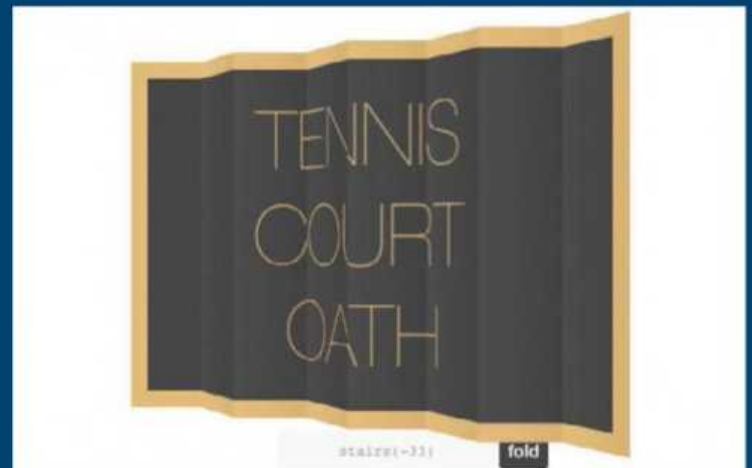
Getting started with OriDomi is accomplished by creating a new instance of the OriDomi class. It can take a pointer to an object, but is also able to process a DOM selector:

```
var folded = new
OriDomi(document.
```

```
getElementsByName('paper')
[0]);
```

Your instance can then be customised. Animations can be tied into your application by the passing-in of callback functions. They get invoked as an animation step is complete: you can use them to fire off a new animation, or can also take them as impetus for more significant changes in the DOM.

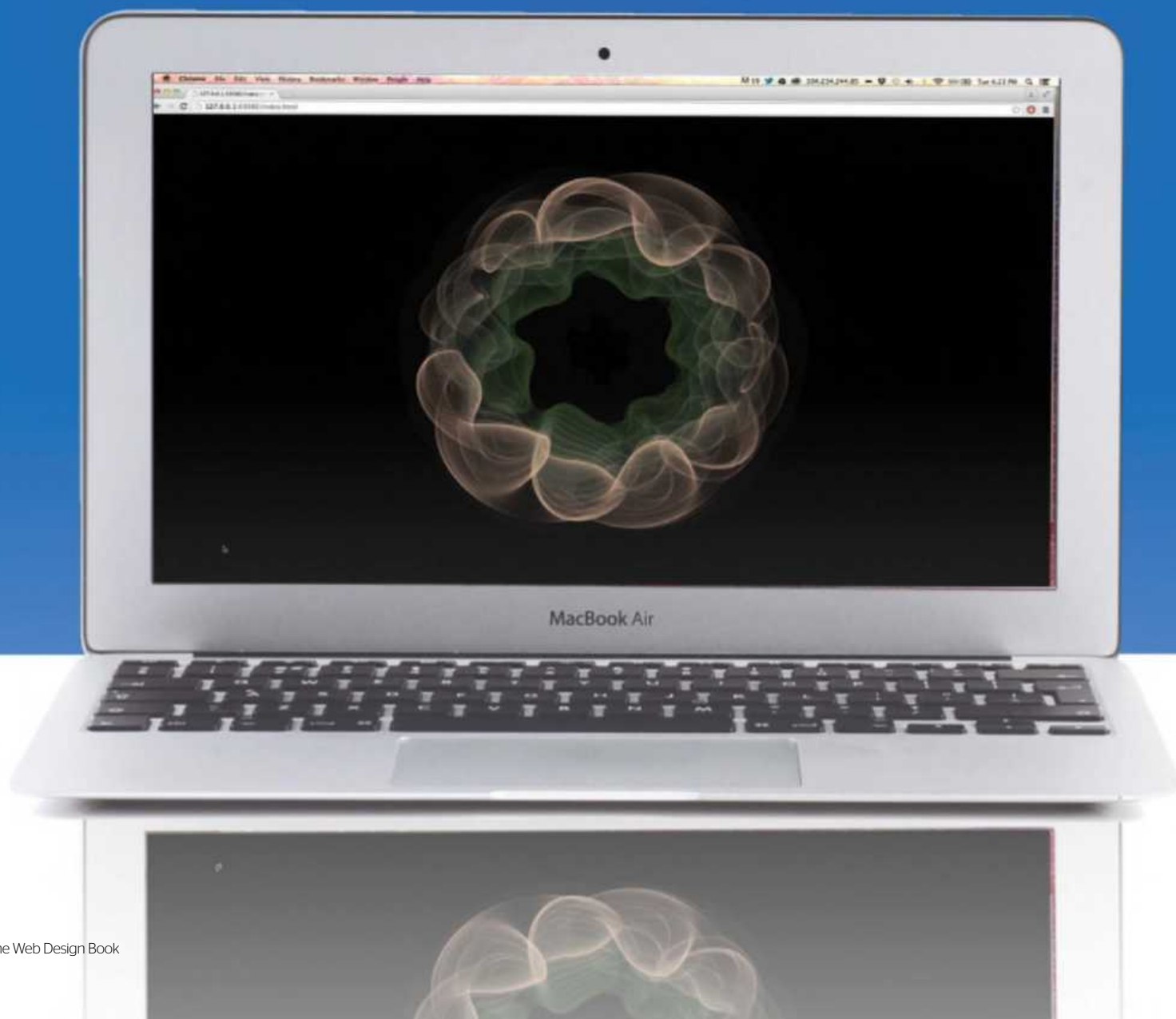
OriDomi uses CSS3 transforms. The plugin will not work on IE10 and older. OriDomi.isSupported can be invoked in order to find out whether your code is running in a supported browser.



↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Make dynamic graphics with the p5.js library

Create interactive drawings with HTML5 Canvas and p5.js, a JavaScript port of the popular open source Processing library





Since the early 2000s, the Processing library has been used by designers and artists as a creative coding environment to create stunning visuals with very few lines of code.

The Processing library is actually a Java library and this is useful for creating interactive installations and prints. The p5.js project is a new initiative to bring the ease of Processing to the web by enabling designers and developers the same easy controls to draw to the Canvas element in HTML5. This means that live, interactive visuals can easily be created for just about any HTML5 capable browser, which is almost any browser from the last three or four years.

The screenshots for this project do not do it justice as the project has to be interacted with in order to see it work properly. As the user moves their mouse, the shape ripples and changes to create a stunning interactive shape-shifting visual. With a little manipulation this could easily become a music visualiser!

1. Start the project

In your web browser, type in p5js.org, click Download and then choose Complete Library. Open the folder 'empty-example' and copy the file p5.min.js into it. Open index.html in your code editor and change the script file to use the minified version.

```
<script language="javascript" type="text/
javascript" src="p5.min.js"></script>
```

2. Open sketch.js

The main code for our project will be found in sketch.js - this is a bare-bones template ready for us to start with the coding process. Open this up in your code editor and you will see that there are two functions here. The first

function, setup, is used to set all the things at the start of the project like background colour and size. The draw function is called every frame.

```
function setup() {
  // put setup code here
}
function draw() {
  // put drawing code here
}
```

3. Edit the setup function

In the setup function add the code that is presented below. This creates a new Canvas element to draw into and set this up to be the current width and height of the browser window. The background of the Canvas is set to black, and the Canvas itself is set to smooth. This will ensure that the lines are crisp at the edges.

```
createCanvas(windowWidth, windowHeight);
background(0);
smooth();
```

4. A new function

In between the setup and draw function, add a new function as we have shown in this step. This code is called to draw an ellipse on the screen and we will be drawing a large number of ellipses later using this. The fill colour is turned off and the stroke is white but with a low opacity that is just over 10 per cent visible. The ellipse is drawn from 0, 0 pixels to 120, and 80 pixels on the x and y axis respectively.

```
function drawEllipse() {
  noFill();
  stroke(255, 255, 255, 28);
```

```
ellipse(0, 0, 120, 80);
}
```

5. Start the draw function

Now inside the draw function, add the first line in the code below. The difference here is that it is called every frame and therefore removes any drawing from the previous frame so that we end up with a new drawing every frame. The translate line moves the drawing point to the middle of the browser window.

```
background(0);
translate(windowWidth/2, windowHeight/2);
```

6. Draw around a circle

A 'for' loop is added here. The 'for' loop increases by 0.5 every loop so it will draw 720 instances. The push line enables it to rotate around. The drawing position is then pushed out 200 pixels on the y axis.

```
for (var i=0; i<360; i += 0.5) {
  push();
  rotate(radians(i));
  translate(0, 200);
```

7. Rotate again

Now as the drawing point is 200 pixels out from the

Adding colour in p5.js

Colour in p5.js follows a pattern of 0-255 to give 256 colours for each colour channel and this in turn will give around 16 million different colours! To get random colours, a random number in that scale can be created.

p5.js

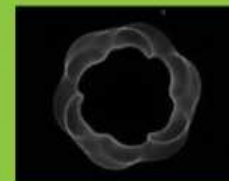
Download • Start • Reference • Libraries • Learn • Contribute

Hello! p5.js is a JavaScript library that starts with the original goal of Processing, to make coding accessible for artists, designers, educators, and beginners, and reinterprets this for today's web.

Using the original metaphor of a software sketchbook, p5.js has a full set of drawing functionality. However, you're not limited to your drawing canvas, you can think of your whole browser page as your sketch! For this, p5.js has add-on libraries that make it easy to interact with other HTML5 objects, including text, input, video, webcam, and sound.

p5.js is a new interpretation, not an emulation or port, and it is in active development. An official editing environment is coming soon, as well as many more features!

p5.js is developed by a group of collaborators, with support from Processing Foundation and NYU/ITP.
© Info: Updated April 6, 2015.



<Left>

- The p5.js library can be downloaded from p5js.org where all documentation can also be found as well as some very useful example files to introduce you to some concepts. Also look out for the p5.js editor, available on Mac OS X and coming soon to Windows and Linux

<Top left>

- Once the core code is in place it can be run and you will see a circular pattern appear on the screen. Next we will make this modulate and change based on mouse position so that it is interactive with the user

<Top right>

- By adding some minimum and maximum values to the waves undulating around the shape, we can start to affect the interaction of the graphic in much more detail than we were able to previously

jQuery & JavaScript

centre of the screen, p5 will draw an orbit around that point but first it will rotate again here. The whole drawing will be scaled but it will also be based on a sine wave that modulates between 1 and -1.

```
rotate(radians(i*3));
scale(map(sin(radians(i*6)), - 1, 1, 0.5,
1), map(sin(radians(i*3)), - 1, 1, 0.5,
1));
```

8. Draw the ellipse

Now as the drawing point is at the right place, the ellipse is drawn and the pop command places the drawing point back to the centre of the document. Save and view the index.html page in the browser to see the drawing of the organic shape in action. Now we change the shape interactively based on the mouse movement of the user.

```
drawEllipse();
pop();
}
```

9. Add some variables

To add interactivity, we will need some variables to dynamically control some of the numbers that draw the shape. Here the variables are added just above the setup function so that we can call them from any function that

we want. The names are fairly self-explanatory, and you will also see them in action in the coming steps.

```
var scaleX;
var scaleY;
var min;
var max;
var radius;
var r, g, b;
```

10. Scale based on position

In the draw function add the following code at the top, and this code will set the values of scaleX to take the mouse's x position. As it moves 0 pixels to the right-hand side of the screen, those values will be mapped onto a range of values from 1.5 to 11.5. The same thing will then happen with the y position of the mouse and the window height as well.

```
scaleX = map(mouseX, 0, windowWidth, 1.5,
11.5);
scaleY = map(mouseY, 0, windowHeight, 1.5,
11.5);
```

11. Amend the 'for' scale

Look inside the 'for' loop, further down in the draw function. Here you will see a line of code beginning with 'scale'. Change the numbers from 6 and 3 to scaleX and scaleY as we have shown here. This means that the code will now take the mouse position for how the shape will be drawn. Save this now and try it out in your browser to see it in action.

```
scale(map(sin(radians(i*scaleX)), - 1, 1,
0.5, 1), map(sin(radians(i*scaleY)), - 1, 1,
0.5, 1));
```

12. Map other values

Just below the code that was added in Step 10, add the code that we have included below. It looks very similar to our other pieces of code, but what's different is that this time the position of the mouse on the x axis is mapped onto a range of numbers from just 0.1 to 0.5 and the y mouse is mapped onto 0.8 to 1.8. These positions will provide some subtle differences.

```
min = map(mouseX, 0, windowWidth, 0.1,
0.5);
max = map(mouseY, 0, windowHeight, 0.8,
1.8);
```

13. Scale again

Locate the 'scale' line of code inside the 'for' loop. Here the min and max variables are added. Save this and view it in the browser to see how it adjusts the drawing on the screen. Remember that these numbers will be adjusted as the mouse moves around on the screen.

```
scale(map(sin(radians(i*scaleX)), - 1, 1,
min, max), map(sin(radians(i*scaleY)), - 1,
1, min, max));
```

14. Change colours

In much the same way as the shape of our graphic can be altered based on where the mouse is, as we have shown, it is also possible to dynamically change the colour. In the setup function add these random numbers for the red, green and blue values. The colour values are measured from 0 (for darkest) to 255 (the brightest).

```
r = random(255);
g = random(255);
b = random(255);
```

The nature of code

Because p5.js is a port of Processing, most Processing code can quickly be adapted. Check out Dan Shiffman's eBook *The Nature of Code* (natureofcode.com/book) for more information.



<Top left>

- A random colour is generated each time the browser is refreshed, this works well at setting an initial colour but what we really want to do is give the user a little more control over the colour that appears rather than giving them sporadic choices

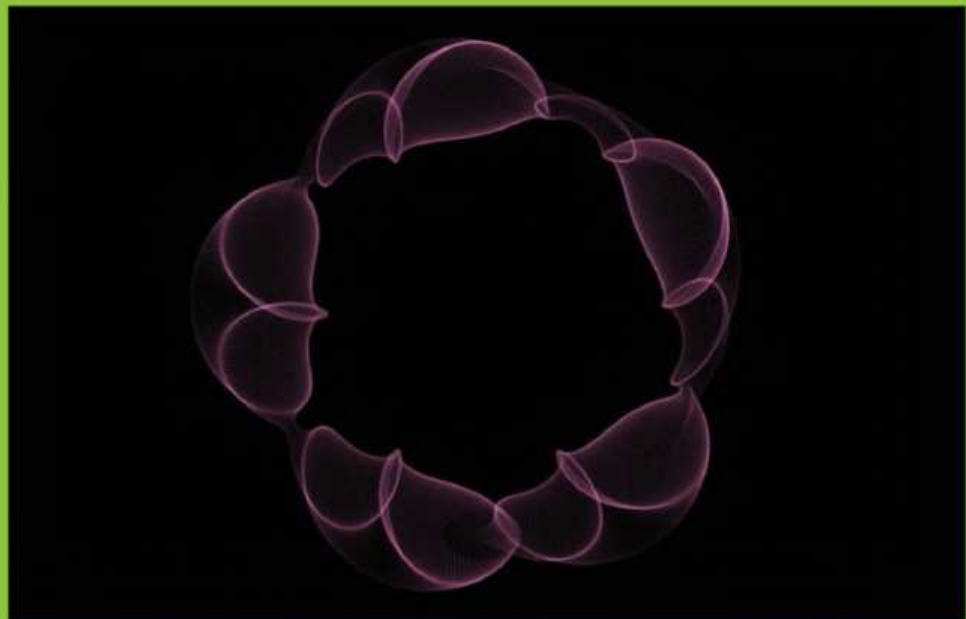


<Top right>

- The colour is now adjusted so the height of the mouse in each third of the screen controls either red, green or blue and this provides a way of adding slightly more control over the colour on the screen

<Right>

- Now the radius of the overall shape has been changed so that it is affected by the mouse position on the y axis. The top of the screen makes the object smaller while the bottom of the screen makes it larger



Getting to know hello p5.js

Starting with a new library can take a little getting used to and a lot of programming languages seem to start with a simple 'hello world'. Taking this idea further, p5.js created a site called 'hello p5.js' (hello.p5js.org), which introduces the basic ideas in a very simple interactive video format. Live code can be seen running alongside the video as the presenters introduce the video. The video itself even has its green screen around the presenters removed live by code in the browser so that the power of what is available through p5.js can be clearly demonstrated. In the video you will see how a particle system is created, a flocking behaviour is added to it and then live windspeed data from New York affects the flocking!



15. Use the colour

To get those random colours working add the letters r, g and b to the stroke line which is located in the draw Ellipse function. This means that when it draws the ellipse it will also make use of the random values. Save this now and test it in the browser to see how it works and looks. Each time you refresh, a new colour will be used.

```
stroke(r, g, b, 28);
```

16. Colourful mouse moves

We may also want to update the colour based on the screen position of the mouse. This colour effect is possible via an 'if' statement as we have shown in the code below, which divides the screen into three horizontal strips for RGB colour. In this step we have started off with the first third and mapped it to the red, giving it a random colour.

```
function mouseMoved() {
  if (mouseX < (windowWidth/3)){
    r = random(255);
    //r = map(mouseY, 0, windowHeight, 0, 255);
  }
```

17. Green and blue changes

We can use the 'if' statement to take the last third of the screen and map that to a random blue colour. Then, we can also do the same for the green in the middle. Save and test it in the browser to see all of the random colours as the mouse moves around, all over the screen. Keep in mind though that each third will affect a different colour.

```
else if (mouseX > ((windowWidth/3)-
  windowWidth)){
    b = random(255);
    //b = map(mouseY, 0, windowHeight, 0, 255);
  } else {
    g = random(255);
    //g = map(mouseY, 0, windowHeight, 0, 255);
  }
}
```

18. Slight change to the colour

We can also specify that the colour changes specifically based on how far down the page the mouse is. We will need to revisit the 'if' statements that we have already used in Steps 16 and 17. First, comment out the random line and uncomment the line afterwards. Then, save this and test this change in the browser to see it all working.

```
//r = random(255);
r = map(mouseY, 0, windowHeight, 0, 255);
```

19. Change the drawing radius

Itself, it is possible to change how large the circle is drawn around the centre point. In the setup function add the radius value as shown here. This started off the project with an initial value of 150 pixels as the radius from the centre of the screen, but this can now be updated.

```
radius = 150;
```

20. Update the radius

Now in the mouseMoved function, add the following line

just before the closing bracket of the function. The y position of the mouse will decide how large the radius of the circle will be. Note that this will be between 100 pixels and 350 pixels, but you can change these values if you want them to be slightly larger or smaller.

```
radius = map(mouseY, 0, windowHeight, 100,
  350);
```

21. Set the radius

For these radius values to actually be used, look inside the draw function and inside the 'for' loop is a translate line. Update the value of 200 to the radius variable. Now you can save the file and then refresh in the browser to see this in action. There will be a lot of changes now to the shape, size and colour.

```
translate(0, radius);
```

22. Fade out

Finally it is possible to fade out the drawing from all of the previous frames. In the 'draw' function, comment out the background colour. So now we want to set the fill colour to black with a low opacity of around 10 per cent. Then, a rectangle is drawn over previous frames with the low opacity on top and this will cause the previous frames to fade out. Save and test this now to see the final effects in all their glory.

```
background(0);
fill(0, 25);
rect(0, 0, windowWidth, windowHeight);
```

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Create shuffling text effects with jQuery

inspiration **s5-style.com**

Portfolio websites for web designers and front-end developers are a really great place to show off their own work. With no clients to dictate the look, content or styling of the site, it presents the designer with an opportunity to carve out their own unique style. This has been the case for Shogo Tabuchi who has created a CSS animated masterpiece with **s5-style.com**.

This website has everything from jittery, glitch-like text effects to 3D background panels that move to new locations every five seconds or so. Even the menu unfolds and animates in a very stylish way to reveal the menu items on the right-hand side.

Here we show you how to recreate this effect, don't forget to head to www.filesilo.co.uk/bks-747 for the tutorial files.



<comment>
What our experts think of the site

Noisy minimalism

"I have tried to keep a careful balance between a glitchy design and a minimalistic design – what I call 'noisy minimalism'. I used HTML5 PushState API to implement seamless image transitions when users click on navigation links and expressive animations on mouseover. I chose GreenSock as it's lighter than jQuery and let me create a wider range of 3D animation with CSS3." **Shogo Tabuchi**

Music on websites

S5 Style has a background music track that plays automatically, this can be annoying for some visitors or they may be in an environment where it isn't appropriate to play sound. As such, always make sure there is a way to mute the sound on the site.

Jumbled text

The "Hello world" text in the centre of the page has a shuffling text effect that changes each of its letters from the initially untangible collection into readable words.

Music feature

The site music can be turned off with a handy button in the top-right of the interface. There is also a button that provides more information about the musician and links to his SoundCloud too.

3D canvas

The background contains animated lines on canvas elements that are displayed in 3D space. These lines can be clicked and will lead to their own individual sections.

Layout shapes

Every few seconds the background tiles move to a new location in 3D space, forming different layouts such as a grid, a circular layout and a long line.

Folding menu

The menu has a prominent position which makes it very unique. Clicking this causes the menu to fold in from the middle of the right-hand side of the screen.

Inspiration

Background animation

The panels that appear in the background to [s5-style.com](http://www.s5-style.com) are beautifully crafted in code. Each panel is a canvas element and each of the panels have what appears to be a blur of lines on them. This is actually a single column of pixels from the image of

that section of the portfolio. As the image appears on the screen the single column moves along the image, producing an animation of lines that are stretched beyond recognition.

This type of effect is known as a slit-scan effect and it first came to prominence in the end sequences of the iconic Stanley Kubrick film *2001: A Space Odyssey*.

Technique

Make the jittery text

1. Body content

In the body section of the page add some content to work with. There needs to be a div on the outside, which is named 'wrapper' and a div on the inside that has been named 'jitter'. Notice we also have a link as this makes a great rollover effect for text.

```
<div class="wrapper">
  <div class="jitter"><a href="#"
    id="text">Hello World</a></div>
</div>
```

2. Link the code libraries

In the head section of the page, add in links to jQuery and to the locally hosted Character Cycle plugin that is available for download from <https://github.com/robinwillis/CharCycle>. It's also included on FileSilo.

```
<script type="text/javascript"
  src="http://ajax.googleapis.com/ajax/
  libs/jquery/1.6.2/jquery.min.js"></script>
<script type="text/javascript"
  src="jquery.charCycle.0.0.1.js"></script>
```

3. Add jQuery

Now while still in the head section of the document add the script tags and jQuery code that will check when the document is ready. This will then start the jitter effect on the text when the mouse is hovering over the text.

```
<script>
$(document).ready(function(){
```

```
$('.jitter').mouseenter(function(){
  if($(this).hasClass('cycling')==false){
    $(this).charcycle({'target':'#text'});
  }
});
});
</script>
```

4. Style the basics

Move on to creating the stylesheet now and this can be done either in an external stylesheet or in the head section. The page is given a black background with both the body and 'wrapper' div being made to fit the 100% height of the browser.

```
<style>
html, body{ background-color:#000; color:
  #fff; height: 100%;}
.wrapper{ height: 100%;}
}
```

5. Style the content

The final styling makes the jitter centrally aligned inside the wrapper div element. Here the link text is made a little bigger, while having a white, italic styling applied to it. Save the page now and view it in the browser, roll your mouse over the text to see the effect in action.

```
.jitter{
  height: 100px; position: relative;
  top: 50%; transform: translateY(-50%);

  text-align: center;
}
a {color: #fff; text-decoration:none;
  font-style: italic; font-size: 60px;}
</style>
```

Hello; r15..

<Above>

- The jittery text effect of the www.s5-style.com site can be reproduced quite easily thanks to the handy CharCycle jQuery plugin.

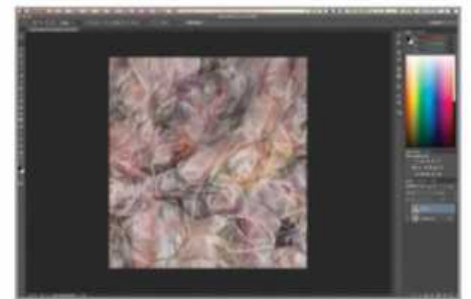
Technique

Create moving lines

In the website there are a number of moving line backgrounds - these have been produced using the canvas element, however it is possible to recreate this using a simple animated GIF technique.

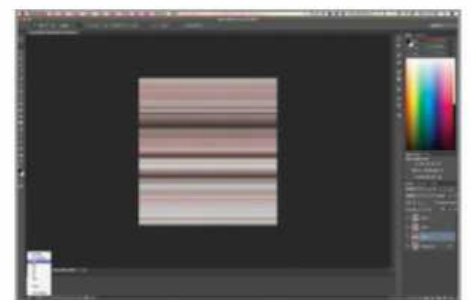
1. Create selections

In Photoshop, use the single column marquee tool to grab a section of the image. Then copy and paste this section to a new layer and scale this horizontally to fill the image. Repeat this until you have several layers each slightly different.



2. Make the animation

Go to Window>Timeline and in the new panel that opens, click on 'Create frame by frame timeline'. Under the frame, click the down arrow and tell the frame to hold for 0.2 seconds.



3. Trim the artboard

Now duplicate the frame and turn off the visibility of the top layer. Repeat this step until you have cycled through each frame. Now go to File>Save for Web and choose GIF as the file format.



Design a 2D game using the Pixi engine

Use the WebGL renderer to make simple, browser-friendly, multiplatform games **tools | tech | trends** Pixi.js



o you're a JavaScript developer and you want to dip your toe into the world of gaming? Well, this is for you!

As a heads up, this tutorial is going to assume a few things. It assumes you know a little about object-oriented programming and that this is not your first foray into the world of JavaScript!

The game we are going to be making is called Flap Pixie Flap – essentially a *Flappy Bird* clone. As it turns out, *Flappy Bird* is perfect for introducing the basic structure and flow of a game.

We will go through and create three classes: a Player class, a Pipe class and finally a Game class and we will use them to create our little game. We are going to use Pixi.js for the rendering, and the WebGL and Canvas 2D

renderer can then be used to take care of it. We will also be using pure OOP JavaScript for the rest and focus on the game structure and let Pixi take care of putting graphics on screen.

This tutorial will show you the basics of making a game, but there is plenty of scope for adding little touches that can take things to the next level. Once you've finished, why not develop it some more!

1. Create Steve (our pixie player!)

The first thing to be done is we create Steve who will be our main protagonist (yes that's his real name!). This Class extends the Pixi MovieClip Object. We also need to give the class a few new variables that will enable Steve to fly and flap!

```
var Steve = function(){
  var frames = [
    PIXI.Texture.fromImage('assets/
    characterFlying_01.png'),
    PIXI.Texture.fromImage('assets/
    characterFlying_02.png'),
    PIXI.Texture.fromImage('assets/
    characterFlying_03.png')
  ];
  PIXI.MovieClip.call(this, frames);
  this.anchor.set(0.5);
  this.speed = new PIXI.Point();
  this.gravity = 0.4;
  this.maxSpeed = 10;
  this.position.x = 240;
  this.spinSpeed = 0;
  this.play();
  this.animationSpeed = 0.4;
}
Steve.prototype = Object.create( PIXI.
MovieClip.prototype );
```

2. Make him fly!

Now we give Steve the flap ability. When called, this will give him a nice speed boost of 15 pixels, shooting him up

into the air. The update function will apply gravity so this speed boost will slowly disappear until he actually starts falling again.

```
Steve.prototype.flap = function(){
  this.speed.y += 15;
}
```

3. Make him hit

When Steve hits something we are going to want to change his state to hit mode – this will mean he animates in a slightly different manner. In this case we make him spin by setting his spin speed!

```
Steve.prototype.hit = function(){
  this.speed.y += 15;
  this.spinSpeed = 0.1;
}
```

4. Create a reset function

Finally we need Steve to be reset. As with all game elements, it's essential to be able to get your object back to its original starting state. In this case we are simply setting Steve's position to 200, resetting his rotation and bringing his speed back to zero.

```
Steve.prototype.reset =
function(){
  this.position.y = 200;
  this.speed.y = 0;
  this.spinSpeed = 0;
  this.rotation = 0;
}
```

5. Make sure he updates

Next we create an update function. This will be called on each game frame and will control the movement of Steve. In this case we want his position to be incremented by his speed each frame. We also want to apply gravity to speed so that eventually he will fall. Now cap his speed to keep him playable.

```
Steve.prototype.update = function(){
  @this.speed.y += this.gravity;
  this.speed.y = Math.min(this.speed.y, this.
  maxSpeed);
  this.speed.y = Math.max(this.speed.y,
  -this.maxSpeed);
  this.position.y += this.speed.y;
  this.rotation += this.spinSpeed;
}
```

6. Piping hot!

The second important component of our flapping game is to have some pipes that need to be avoided. The Pipe class extends a Pixi container object and we add to that a Pixi sprite to represent the top pipe and another for the bottom one.

```
var Pipe = function( entryPoint, maxHeight,
minHeight ){
  PIXI.DisplayObjectContainer.call(this)
  this.entryPoint = entryPoint;
  this.maxHeight = maxHeight;
  this.minHeight = minHeight;
  this.gapSize = 300;
  this.topPipe = PIXI.Sprite.
```



<Top left>

- Flap Pixie Flap in all its glory! It's a small but fun little game that should provide the perfect entry into the world of HTML5 game programming

<Top right>

- Adding a trail will give the game a sense of speed. Little touches like this are key to elevating your game above the rest!

<Bottom left>

- Steve makes use of Pixi.js' MovieClip class and uses three frames of animation to flap his little wings

<Bottom right>

- This lush jungle background is the perfect setting for Flap Pixie. It makes use of Pixi.js' TilingTexture class



```
fromImage('assets/column.png');
this.bottomPipe = PIXI.Sprite.
fromImage('assets/column.png');
this.addChild(this.topPipe);
this.addChild(this.bottomPipe);
this.adjustGapPosition();
}
```

7. Make pipe adjust its position

Here we give the pipe the ability to move its gap. The game would be pretty boring if all the gaps were in the same place! Our code generates a random number between the min and max positions that are based on the positions of the top and bottom pipe accordingly, creating a gap that equals the gapSize property.

```
Pipe.prototype.adjustGapPosition =
function(){
this.gapPosition = this.minHeight + ( Math.
random() * (this.maxHeight - this.minHeight)
);
this.topPipe.position.y = this.gapPosition
- this.gapSize/2 - this.topPipe.height;
this.bottomPipe.position.y = this.
gapPosition + this.gapSize/2;
}
```

8. Make our pipe update

This is the pipe's update function that needs to be called each time the game updates. A pipe basically needs to move across the screen each frame. Once the pipe has left the screen it then needs to reposition itself on the other side of the screen (like a pipe conveyor belt!). At

that point, we then also tell the pipe to adjust its gap position to make the game interesting!

```
Pipe.prototype.update = function( speedX ){
this.position.x -= speedX;
if(this.position.x < -200){
this.position.x += this.entryPoint;
this.adjustGapPosition();
}
}
```

9. Time to put the two together

Here we create our game class. This will create and manage the game elements for us. First we set our game variables then initialise all the game objects. We also add a mousedown and a touchstart callback to the stage so we can react when a user clicks or touches the screen. Then reset the game and start the update loop.

```
var Game = function(){
this.width = 1286;
this.height = 640;
this.gameSpeed = 5;
this.pipes = [];

this.state = 'playing';
this.initPixi();
this.initPipes();
this.stage.mousedown = this.stage.
touchstart = this.onClicked.bind(this);
this.steve = new Steve();
this.stage.addChild(this.steve);
this.reset();
}
```

Pretty as a picture!

Games with good code and good design are greater than the sum of their parts. Developer graphics are 'unique' but stick to what you're good at and get a designer buddy to help out with the art.

```
requestAnimationFrame(this.update.
bind(this));
}
```

10. Set up Pixi

Games tend to be more fun when you can see them huh? This function creates a Pixi renderer and adds to the web page. Pixi will automatically decide if it should use a WebGL or Canvas renderer for you (nice!). We then create a nice background and add that to our stage too. Finally we make the stage interactive so we can click it!

```
Game.prototype.initPixi = function(){
this.stage = new PIXI.Stage(0x66FF99);
this.renderer = PIXI.
autoDetectRenderer(this.width, this.height);
document.body.appendChild(this.renderer.
view);
this.background = new PIXI.
TilingSprite(PIXI.Texture.fromImage('assets/
mainBG.jpg'), this.width, this.height);
this.stage.addChild(this.background);
this.stage.interactive =
true;
this.stage.hitArea =
```

jQuery & JavaScript



<Left>

• It's all about getting the balance just right! This takes time and effort, but it will always pay off

<Right>

• For most games simple hit testing provides more than enough accuracy, whilst still offering decent performance

Game engines

Knowing how to make a game from scratch will certainly make you a better developer in the long run. But remember there are lots of HTML5 game engines out there that will give you a head start too!

```
new PIXI.Rectangle(0, 0, this.width, this.height);
}
```

11. Set up the Pipes

Time to set up the Pipes! This function basically creates eight new Pipes and spreads them out with a 200 pixel gap. We also pass the Pipes the information they need to be able to take care of their scrolling. Add them to the game stage and push them into an array so we can access them later.

```
Game.prototype.initPipes = function() {
  var pipeWidth = 139;
  var pipeGap = 200;
  var totalPipes = 8;
  var size = (pipeWidth + pipeGap) * totalPipes;
  for (var i = 0; i < totalPipes; i++)
  {
    var pipe = new Pipe( size, 200, this.height - 200 );
    this.stage.addChild(pipe);
    this.pipes.push(pipe);
  };
}
```

12. Call onClicked

This gets called every time the user clicks or taps the screen. First check the game state. If the game is in 'playing' mode then call the flap function on Steve. This will make him fly up a little. Otherwise we know that the state is in game over and our little guy has perished, and in that case we call the reset function.

```
Game.prototype.onClicked = function(){
  if(this.state === 'playing'){
    this.steve.flap();
  }
}
```

```
}
else{
  this.reset();
}
}
```

13. Add some hit testing for x

Without hit testing, we don't have a game! This function takes a pipe and then hit tests it against Steve, returning the result. First check if Steve's x position is further than the pipe's x position. In the same line we then check to see if Steve has passed the pipe's x position plus its width. If this line is true then Steve is going through a pipe. The next step is to determine if he is going through the gap or hitting the pipe.

```
Game.prototype.hitTestPipe = function( pipe ){
  var playerHitArea = this.steve;
  if( playerHitArea.x + playerHitArea.width/2 > pipe.position.x &&
    playerHitArea.x - playerHitArea.width/2 < pipe.position.x + pipe.width){
```

14. Hit test the y

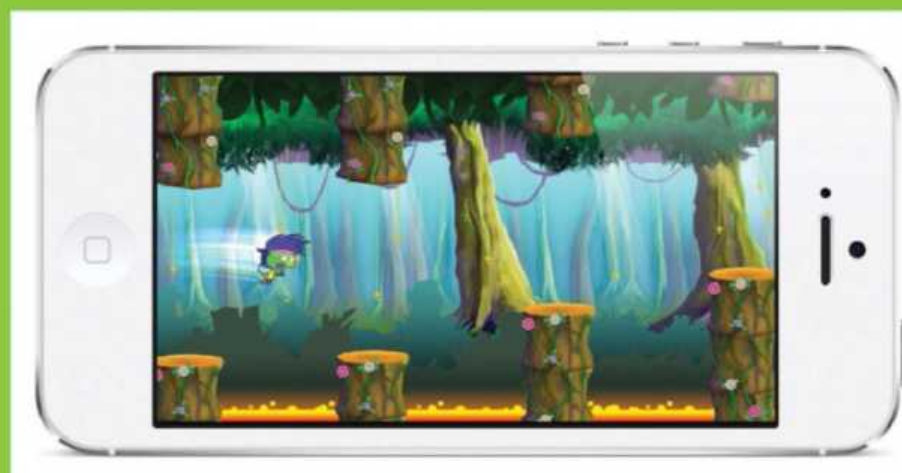
Now look at the y positions. Check if the top of Steve is above the bottom of the top pipe. If true, check if the bottom of him is lower than the top of the bottom pipe. If either of these are true then Steve is touching a pipe. Return a Boolean indicating any collisions.

```
if( playerHitArea.y - playerHitArea.height/2 < pipe.topPipe.position.y + pipe.topPipe.height ||
  playerHitArea.y + playerHitArea.height/2 > pipe.bottomPipe.position.y){
  return true;
}
return false;
}
```

15. Game over!

A game must end. When this function is called we change the game state to 'gameover' and also call hit on Steve so that he can react to colliding with the pipe.

```
Game.prototype.gameover = function(){
```



<Above>

• The great thing about Flap Pixi Flap and other HTML5 games is that they immediately run on pretty much all mobile browsers



Balancing your game, find the fun!

The key aspect of any game is to make it fun. This is done through the process of balancing a game by tweaking all of its various properties until a game feels fun and enjoyable. This process often falls to the game designer, but more often than not there may not be one on your team, so it's down to you. The key to balancing a game is to simply do it! It's a simple process - tweak your game properties, play it, ask 'is it fun yet?' Then rinse and repeat until the game feels right. Patience and persistence are key here. Taking the time to do this properly (and trust me it takes longer than you think!) will make the difference between an okay game and a great game.

```
this.state = 'gameover';
this.steve.hit();

}
```

16. The reset function

One of the most important functions! A game needs the ability to reset itself so the player can play the game again. In our case we set the game state back to playing, reset Steve and readjust all of the pipes.

```
Game.prototype.reset = function(){
  this.state = 'playing';
  var pipeWidth = 139;
  var pipeGap = 200;
  var totalPipes = this.pipes.length;

  for (var i = 0; i < totalPipes; i++){
    var pipe = this.pipes[i];
    pipe.position.x = ((pipeWidth + pipeGap) *
    i) + 800;
    pipe.adjustGapPosition();
  };
  this.steve.
  reset();
}
}
```

17. Get the game ticking over

Now for the update loop! First check if the game is in a playing state. If it is, update the background and loop through the pipes, update them and hit test against Steve. Check if Steve has dropped below the screen too.

```
Game.prototype.update = function()
{
  if(this.state === 'playing'){
    this.background.tilePosition.x -= this.
    gameSpeed * 0.6;
    this.steve.alpha = 1;
    for (var i = 0; i < this.pipes.length; i++)
    {
      var pipe = this.pipes[i];
      pipe.update( this.gameSpeed );
      var hit = this.hitTestPipe(pipe);
      if(hit){
        this.gameover();
        break;
      };
      if(this.steve.position.y > this.height){
        this.gameover();
      }
    }
  }
}
```

18. Update Steve

Next update Steve. We do this regardless of the game state, this way he keeps falling when he has hit something (dramatic!). Finally we render our game using the Pixi renderer and then requestAnimationFrame. This keeps the game update to be called every frame (60fps).

19. Load it all up!

So the final part! Let's create a Pixi Asset Loader and pass in an array of image URLs that we are going to load. Next we add and attach an onComplete callback to the loader. This is called when all assets have loaded - the perfect time to create an instance of our game! Finally call the load function to begin the whole process.

Add a little extra

1. Create the Trail class

We have created a new class that contains 20 particles. It extends a Pixi DisplayObjectContainer so it means we can add it to our stage:

```
Trail = function(target){
  PIXI.DisplayObjectContainer.
  call( this );
  this.target = target;
  this.particles = [];
  var total = 20;
  for (var i = 0; i < total; i++) {
    var particle = new PIXI.
    Sprite.fromImage('assets/
    flyingPixie.png');
    particle.life = (i / total-1) * 100;
    particle.anchor.set(0.5);
    particle.blendMode = PIXI.
    blendModes.ADD
    this.particles.push(particle);
    this.addChild(particle);
  }; }
  Trail.prototype = Object.create(
  PIXI.DisplayObjectContainer.prototype
  );
```

2. Create an update function.

Here we do some basic particle updating. Each particle has a life property and as this property decays, we modify its Alpha to make it fade out. This creates a cool trailing effect.

```
Trail.prototype.update =
function(speedX){
  for (var i = 0; i < this.
  particles.length; i++){
    {
      var particle = this.
      particles[i];
      if(particle.life < 0)
      {
        particle.life += 100;
        particle.position.set(this.
        target.position.x, this.target.
        position.y); }
      else
      { particle.life -= 5;
        particle.alpha = ( particle.
        life/100) * 0.75;
        particle.x -= speedX
      }
    }; }
  };
```

3. Now plug it in!

Finally we create a new instance of the Trail class in the game constructor, making sure to pass Steve in as our target. Then in the game's update loop we update the trail.

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Sync animations to audio and video with Popcorn.js

Trigger events easily at any timeframe and play back music or video by using Mozilla's Popcorn.js





nce upon a time syncing up music or video with animations, overlays or other content could be done quite easily with Flash, but since the demise of that plugin it's become harder to

achieve this kind of effect. HTML5 has given us a great way to natively play audio or video in the browser but it's still a hassle to try and get something to happen at a key point in your media. Fear not though because Mozilla has come to the rescue with their media library Popcorn.js. This has a very simple premise to enable the developer to code content that can be triggered at various times during the playback of their media. If you've ever seen interactive music videos by Arcade Fire and Rome for example, then this is the sort of library that enables the triggering of new scenes or animations at key points in their music videos.

We are going to have a 3D scene in the background of the page, created in WebGL with three.js. Over the top of that will be some regular DOM content, which will be hidden. Popcorn will be used to load audio, play it and then trigger camera moves in the 3D content.

1. Up and running

Open the start project folder in Brackets or a similar code editor and open index.html. Scroll down to the body tag and add the following div tags for our on-screen messages that will be shown at key points with the music. Copy and paste this code three more times and change the id to two, three and four.

```
<div id="one" class="outer hide">
<div class="middle">
<h1>&ldquo;Globally, loss-related floods
have more than tripled since 1980&rdquo;</
```

```
br><small>Munich Re Insurance</small></h1>
</div>
</div>
```

2. Link to popcorn

Change the text inside the <h1> tag for the copied sections and you can refer to the text in the finished file or add your own. A little further down the body section you will see a bunch of script tags for the 3D animation, add this at the bottom to link to the Popcorn library.

3. Add the music file

Under the div tags from Step 1 and add this HTML5 audio tag to link to the MP3 audio. Later on, this audio tag will be hidden on the screen using CSS and the Popcorn.js library will link to this so that animation can be triggered based on the current time of the song playing.

```
<audio id="myAudio">
<source src="autumn-leaf.mp3" type="audio/
mpeg">
</audio>
```

4. Start styling

In the head tag on the page, add an opening and closing style tag, then add the following CSS into that. Here the body tag gets the padding and margin removed and the right typeface is set for all content. Any content that overflows the page will be set to hidden, as is the audio from the previous step.

```
body {
padding:0;
margin:0;
overflow:hidden;
font-family: 'Oswald', sans-serif;
```

```
color:#ffba00;
}
audio { display:none; }
```

5. CSS animation

Now the code adds some very simple CSS3 animation. The hide class has been applied to all the content in the body section of the page. A transition for the opacity is applied, which sets the length to half a second. When content needs to fade in, the show class can be applied to the div and it will fade in, taking half a second to do so.

```
.hide {
opacity: 0;
transition: opacity .5s ease-in-out;
-moz-transition: opacity .5s ease-in-out;
-webkit-transition: opacity .5s ease-in-out;
}
.show {
opacity: 1;
}
```

6. Position the messages

Any new content to be displayed needs to be shown over the top of this. As such the z-index, which is like the height, is set to be higher than the rest of the page. This section is set to fill the browser width and height, positioned absolutely in the top left.

Familiar terminology

If you've ever worked with audio or video the term 'cue' means to start a piece of media and hence the code to start something happening is also 'cue' in Popcorn.



< Left >

- The camera is called to start animating immediately as the song starts playing, and you will see the 3D scene zoom closer so that the houses are the focus of attention

< Top left >

- The Popcorn.js library can be downloaded from Popcorn.js.org. There are also some examples so that you can understand how to sync events with the audio

< Top right >

- Once the content is all added to the page, the 3D scene is called to display after the music has finished loading

jQuery & JavaScript

```
.outer {
  z-index: 10;
  width: 100%;
  height: 100%;
  position: absolute;
  top: 0; left: 0;
}
```

7. Vertical alignment

Inside the fullscreen div of the outer created in the previous step is another div. The text inside here should appear in the centre of the screen horizontally and vertically. The middle class here will ensure that happens by aligning it on the vertical axis with the page centre.

```
.middle{
  min-height: 100%;
  min-height: 100vh;
  width: 100%;
  display: -webkit-flex;
  display: flex;
  align-items: center;
  -webkit-align-items: center;
}
```

8. Set the headings

Because the background has a number of colours, there is a text shadow on the text to help it stand out against

Additional plugins

There are additional plugins for Popcorn that allow you to run video from hosted sources such as Vimeo, YouTube and SoundCloud. It's even possible to open Google Maps.

the background. The size of the text is also increased and centred horizontally.

```
h1{
  font-size: 3.8em;
  display: inline-block;
  width: 40%;
  margin: 0 auto;
  text-align: center;
  text-shadow: 3px 3px #000;
}
```

9. Wait for the audio

Before the audio is set to play, it is important to ensure that it has fully loaded. Add the code just under the existing script tags on the page. This will load all of the scripts before calling the init function. The init function will show the 3D background, so test that in the browser.

```
<script>
document.addEventListener("DOMContentLoaded", function () {
  init();
}, false);
</script>
```

10. Calling the music

Under the init(); line, add the next line, which will call a function to play the music. This hasn't been created yet so don't run it in a browser.

```
initMusic();
```

11. Link and play

Before the closing script tag in Step 9, add the following function. This links to the audio with the id of myAudio,

which was added way back in Step 3. Once the link is established, the audio is set to play. Save, and load it, the music will start to play.

```
function initMusic(){
  popcorn = Popcorn( "#myAudio" );
  popcorn.play();
}
```

12. Call the first animation

The scene needs a little movement so that the first message can be displayed. Add the line below inside initMusic. This is calling a function on line 111 of the scene.js file if you want to look at it. Save this and refresh your browser and the camera will move forward in the scene.

```
camMove1();
```

13. Cache the divs

To speed up DOM manipulation, variables are created to hold the four div tags that contain text. The 'show' class will be added and removed and this will make the appropriate message fade in or fade out at the right time. Add this code inside the initMusic function.

```
var one = document.getElementById( 'one' );
var two = document.getElementById( 'two' );
var three = document.getElementById( 'three' );
var four = document.getElementById( 'four' );
```

14. Sync to music

All of the remaining code is added before the closing bracket of the initMusic function. To make something happen in time to music, the following code is used. This calls its own function after one second of the music. Test to see the message show after one second.



<Top left>

- As the bass starts in the music, lights in the scene are turned on and off to look like lightning flashing. Popcorn makes it easy to sync the timing of this to the music

<Top right>

- The camera moves forward again and the third message is displayed to show the relevance of the background. This move is timed to coincide with the car driving past

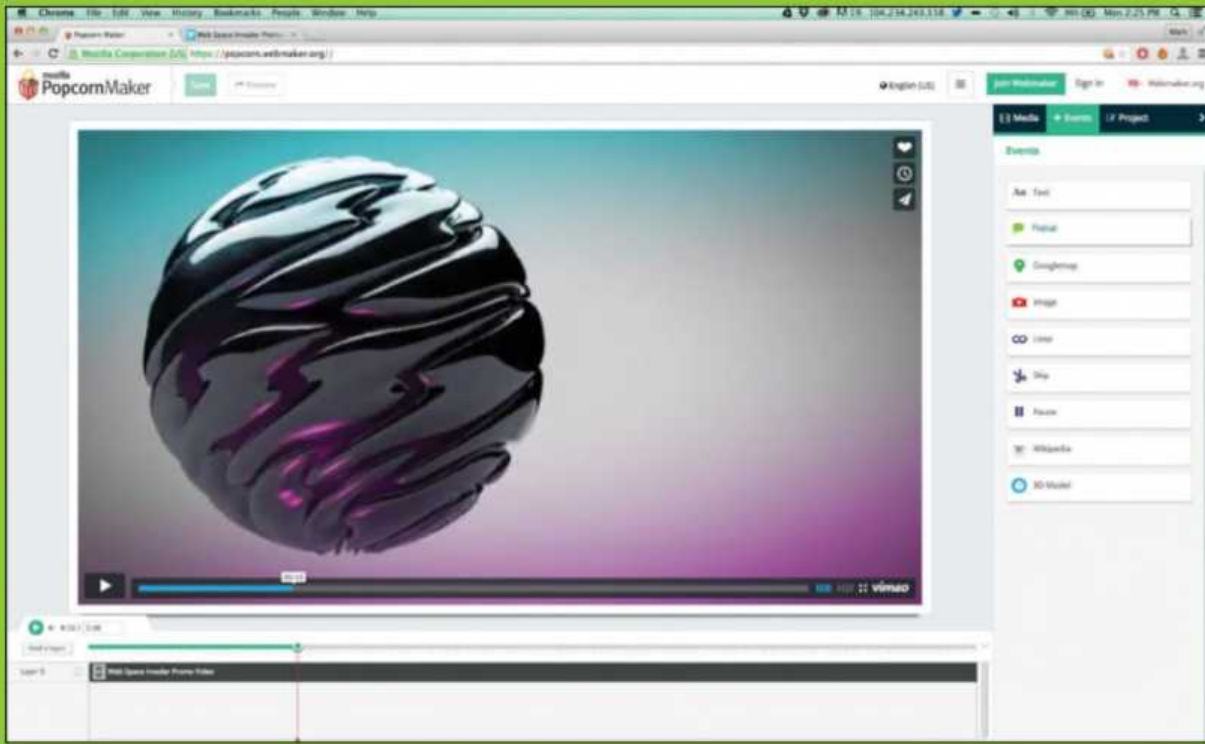
<Right>

- The camera moves forward to focus on the city with the rain animating. As it does this, the second text content is displayed on the screen



Popcorn maker

While Popcorn is easy enough to cue content, with its straightforward JavaScript API, designers might prefer not to get into the code and instead create content for Popcorn with a graphical user interface. Mozilla has made this possible with Popcorn Maker (popcorn.webmaker.org). By loading this page in your browser it is possible to sync up existing content found on YouTube, SoundCloud and Vimeo, to make events happen at different points during the playback of that content. Popcorn Maker comes with a simple timeline so you can scrub through the content to the section when you want something to happen. Adding events lets pop-ups or other content appear. The only problem is that Popcorn Maker is hosted externally to your site.



```
popcorn.cue( 1, function() {
  one.classList.add('show');
});
```

15. Move forward

Adding the next code causes the camera to start to move forward to the next section in the 3D scene by calling `camMove2` inside the `scene.js` file. The text on the screen is made to fade out by removing the `'show'` CSS class, and so the opacity is removed to 0 again.

```
popcorn.cue( 13.3, function() {
  camMove2();
  one.classList.remove('show');
});
```

16. Add the message

Now the next code will show the text in the div with the id of `'two'`. This takes place after almost 16 seconds. Save and test to see the effect.

```
popcorn.cue( 15.8, function() {
  two.classList.add('show');
});
```

17. Lightning flashes

Just after the last message displays on the screen you will hear two bass stabs as part of the music. It's possible to use this with a lightning effect in the 3D scene to make it more dramatic. Here we turn a flash of light on for just less than half a second in time with the first bass stab.

```
popcorn.cue( 17.2, function() {
  lightOn();
});
popcorn.cue( 17.6, function() {
  lightOff();
});
```

18. Slight change to the colour

Let's do the same again on the second bass stab. The code here gets the timing just right for that. You can add more by listening to the audio and noting the times.

```
popcorn.cue( 18, function() {
  lightOn();
});
popcorn.cue( 18.4, function() {
  lightOff();
});
```

19. Continuing the journey

Wait for the song to be almost 28 seconds in to call the camera to move again with `camMove3`. Save and refresh

```
popcorn.cue( 27.7, function() {
  camMove3();
  two.classList.remove('show');
});
```

20. The next message

Just after the camera arrives at the bridge, the next message needs to display on the screen, so in this code the function is called at just after 30 seconds so that the

message displays. You'll notice that an animation of a car drives past at this particular point and this is no coincidence as the message is relevant to that.

```
popcorn.cue( 30.2, function() {
  three.classList.add('show');
});
```

21. Final camera movement

Once the car has driven past the screen it's time to move the camera to the final destination. At just after 40 seconds the final camera move is called, and the third message fades out on the screen by removing the `'show'` CSS class. The camera swings up to the mountain.

```
popcorn.cue( 40.5, function() {
  camMove4();
  three.classList.remove('show');
});
```

22. Finish off

As the camera eases into the final position in the background, the final text is displayed on the screen. At 43 seconds the fourth text block gets the CSS class of `'show'` added so that it fades in, and shortly after this the music finishes playing. Now just save and test the document in your browser to see the full animation and music sync up together.

```
popcorn.cue( 43, function() {
  four.classList.add('show');
});
```

Animate SVGs with the Vivus.js library

Create dynamic high-quality line drawings using Vivus.js and its predefined scripts

tools | tech | trends Your favourite SVG supporting web browser, your favourite text editor, JavaScript, HTML, Vivus.js



Creating graphics in the browser has become thoroughly interesting over the last few years. Cast your mind back to circa 2008, if you wanted to do some generative art or create a graphic on

the fly, you'd probably have to use a plugin like Flash to do it.

Suddenly, from out of the darkness came the HTML5 spec and it contained <canvas> and <svg>, and yay, it was good. Canvas and SVG have matured in the years since they were first implemented in our browsers and absolutely astonishing things have been done with them - whole worlds have been visualised with Canvas and the crispest of scalable graphics are displayed with SVG.

Today, we're going to look at a JavaScript library called Vivus. Vivus.js will enable us to take any SVG element on a HTML page and then animate it line by line to reveal the entire shape. If you're looking to add just a little bit of polish to your own projects, or if you are trying to show off to a favourite client, Vivus is a quick and impressive way of doing just that. So let's start working on it then!

1. Grab Vivus.js

First, we need to grab the Vivus library. Download it as a ZIP file from github.com/maxwellito/vivus and expand the archive. Open the src folder and copy both the vivus.js and pathfinder.js files into your project folder. Then, simply reference these files in your HTML.

```
<code>
...
<script src="/path/to/your/script/
pathfinder.js"></script>
<script src="/path/to/your/script/vivus.
js"></script>
</body>
</code>
```

2. Set up for SVG

In order to animate SVGs, we need some SVGs to start off with! Create and open a simple HTML page and insert the following:

```
<code>
<!DOCTYPE html>
<html>
<head>
<meta content="text/html; charset=utf-8"
http-equiv="Content-Type">
<title>Web Designer Vivus Demo</title>
<link rel="shortcut icon" href="favicon.
ico">
<link rel="stylesheet" href="styles.css"
type="text/css" />
<meta name="viewport" content="initial-
```

```
scale=1.0, user-scalable=no" />
</head>
<body>
<svg id="box">
<polygon points="20,20 40,20 40,40
20,40 20,20" style="stroke:rgb(40,40,40);
stroke-width:1; fill:none;" />
</svg>
<script src="scripts/libraries/
pathfinder.js"></script>
<script src="scripts/libraries/vivus.
js"></script>
</body>
</html>
</code>
```

...and create a new CSS file called styles.css and add the following rule to it:

```
<code>
svg{
width: 200px;
height: 200px;
margin: 10px;
display: block;
position: relative;
margin-left: auto;
margin-right: auto;
}
</code>
```

This will create an SVG element in our page that is 200 by 200 pixels and aligns to the centre of our

page, inside of which there should be a 20 by 20 pixels box.

3. Create the first animation

Now that we have our first quick SVG out of the way, we can make our first quick animation. Just before your </body> tag, create a new <script> tag and add the following code:

```
<code>
...
<script>
new Vivus('box');
</script>
</body>
</code>
```

We've just created a new Vivus object and have passed the ID of the SVG we want to animate to it. If you refresh your page, you should see your box draw itself on your screen.

4. Choose Vivus animations

The code we just ran was the simplest way to get Vivus running. Vivus has a multitude of options that we can pass through as a second argument to any Vivus object. There are three types of animations available for use: delayed, async and oneByOne, which we can choose:

```
<code>
...
<script>
new Vivus('box', {type : "async/ oneByOne/
```


**<Above>**

Vivus can animate almost any shape, providing it's a polygon and not a <rect> or <circle>. We can define the shape by first adding a points attribute to our polygon element

**<Above>**

We can add a style to determine the paths' colours and widths. Try to avoid inline styles in HTML, with SVG it makes sense to indulge rather than create a dozen or so CSS rules

Need for speed?

SVG graphics are graphics defined and drawn using nodes in the DOM. This makes it great for generating graphics on the fly, especially on the server-side. It's this quality, however, that cause its downfall when it comes to animation - manipulating the DOM like Vivus does can be very taxing and thus slower than a <canvas> equivalent render.

```
</code>
```

6. Use the delayed animation

In your script tag, create a new Vivus object, passing through the ID of 'square' instead of 'box'. Now, if you reload your page, you will see our fancy box of diagonal lines draw itself. This is the default delayed animation. This animation starts drawing each line at the same time, but adds a small delay to each path, creating a staggered effect.

```
delayed"));
</script>
</body>
</code>
```

..but in order to appreciate the difference we're going to need a more complex shape.

5. Create a multiple path SVG

To better understand the effects of the different animation types available to us, we're going to need an SVG with multiple paths and shapes. Create a new SVG element and give it an ID of 'square' and add the following points and shapes to it:

```
<code>
...
<svg id="square">

<rect width="200" height="200" style=
"fill:rgba(255,255,255,0); stroke-width:10;
stroke:rgb(40,40,40)"/>
<polygon points="0,40 40,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="0,80 80,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="0,120 120,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="0,160 160,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="0,200 200,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="40,200 200,40"
```

```
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="80,200 200,80"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="120,200 200,120"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="160,200 200,160"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="200,200 200,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="200,160 40,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="200,120 80,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="200,80 160,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
<polygon points="200,40 120,0"
style="stroke:rgb(40,40,40); stroke-
width:6;"/>
</svg>
```

```
<code>
<script>
var second = new Vivus
('square');
</script>
</code>
```

7. Use the async animation

To draw our SVGs with the async animation, we need to pass that option through to the Vivus object for our square.

That's simple enough, all we have to do is pass through an options object with the type defined as our second argument:

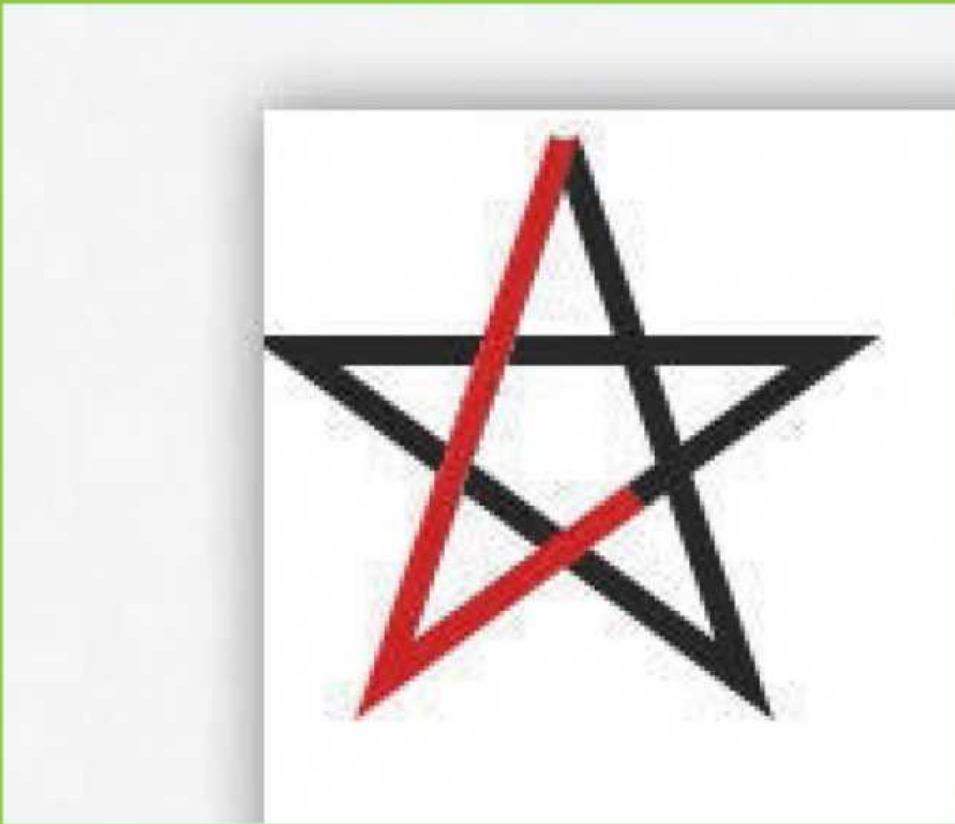
```
<code>
<script>
var second = new Vivus('square', {type :
'async'});
</script>
</code>
```

...the async animation type starts drawing all of the paths at the same time and finishes drawing them later.

8. Use the onebyOne animation

The last of the preset Vivus animations is the oneByOne. Of the three it's the simplest effect, it just draws our SVG's paths as you would expect it to - one by one. To use it, we pass oneByOne through as the type of animation to our Vivus object:

```
<code>
<script>
var second = new Vivus('square', {type :
'oneByOne'});
</script>
</code>
```



<Above>

- Specifically for Vivus, we can add certain other attributes to better control the rendering of our SVG graphic. Data-async is unique in that it doesn't require a value in order to be enacted.

On scroll

You may have observed that without action on the viewers part, our animations play straight away, but that is a mere illusion. The animations in our demo play because they are immediately visible on the page load, those that are off screen will wait until they are scrolled down to. Now you don't have to worry about missing anything.

9. Call the play method

So far, in each of our steps, our shapes have started to animate as soon as our page as loaded. If we want to create custom controls and events for drawing our SVG, we can do that by passing through a 'start' variable in our options object and then calling the play method when we like.

```
<code>
var second = new Vivus('square', {type:
"oneByOne", start : "manual"});
second.el.addEventListener('click',
function(){
second.play();
}, false);
</code>
```

10. Overall time

For each of the animation types, regardless of the number of paths we have to draw, they're all complete after 120 frames. We can use a duration key to determine the frame numbers and change an animation's duration.

```
var second = new Vivus('square', {type:
"oneByOne", start : "manual", duration
:1000});
setTimeout(function(){
second.el.addEventListener('click',
function(){
second.play();
}, false);
}, 1000);
</code>
```

11. Durations and frames

Vivus doesn't use seconds or milliseconds to determine an animation's duration. Each animation has a duration defined in frame numbers, and each line drawn or delay created is defined by that same frame count. If we have a 100-frame long animation, and we delay one of the paths by a value of 20, it will then start to draw a fifth of the full time after the rest of the animation has actually been started.

12. Add a scenario animation

So far, all of our animations have been a variant on a theme without much manoeuvrability for customisations. If we want to fine-tune the events that happen in our animation then we can create a 'scenario'.

A scenario animation will enable us to define durations and delays for each path. Let's now create a new SVG graphic to demonstrate some scenarios. Add the following to your HTML.

```
<code>
<svg id="nightSky">
<polygon points="10,1 4,19 19,8 1,8
16,19" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" />
<polygon points="30,21 24,39 39,28 21,28
36,39" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" />
```

For the full step code see 'SVG-Step12' in the tutorial files on FileSilo.

13. Scenario delays

In your webpage, you'll now have an animation that draws a series of stars in our SVG element. The animation we are seeing is still being determined by the default async animation.

If we add some data-start to our SVG polygons, we can determine a delay before that polygon begins to animate.

```
<code>
<svg id="nightSky">
<polygon points="10,1 4,19 19,8 1,8
16,19" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" data-start="0"/>
<polygon points="30,21 24,39 39,28 21,28
36,39" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" data-start="10"/>
<polygon points="130,121 124,139 139,128
121,128 136,139"
style="fill:none;stroke:rgb(40
,40,40);stroke-width:1;" data-start="20"/>
<polygon points="80,71 74,89 89,78 71,78
86,89" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" data-start="30"/>
<polygon points="80,31 74,49 89,38 71,38
86,49" style="fill:none;stroke:rgb(40,40,40)
;st roke-width:1;" data-start="50"/>
<polygon points="150,51 144,69 159,58
141,58 156,69"
style="fill:none;stroke:rgb(40,4
0,40);stroke-width:1;" data-start="40"/>
<polygon points="150,81 144,99 159,88
141,88 156,99"
style="fill:none;stroke:rgb(40,4
0,40);stroke-width:1;" data-start="20"/>
</svg>
</code>
```

14. Scenario times

Just as we can define when a path should start to be drawn, we can also determine how many frames it should take to do just that. Remember, the fewer the frames, the quicker, but also potentially jumpier the animation is. If there are a greater number of frames, the animation will be smooth but it will also take longer for it to fully complete.

```
<code>
<svg id="nightSky">
  <polygon points="10,1 4,19 19,8 1,8
  16,19" style="fill:none;stroke:rgb(40,40,
  40);stroke-width:1;" data-start="0" data-
  duration="10"/>
  <polygon points="30,21 24,39 39,28
  21,28 36,39" style="fill:none;stroke:r
  gb(40,40,40);stroke-width:1;"
  data-start="10" data- duration="30"/>
</code>
```

For the full step code see 'SVG-Step13' in the tutorial files on FileSilo.

15. Tidy up

With every DOM manipulation, the code we write is not always the code that winds up being viewed in the inspector. Normally that's okay because a state has changed and we need to reflect that. But, in this case we're changing attributes with the aim of creating those that exist in the DOM. If we want to, we can remove all of the extra values created with selfDestroy.

```
<code>
var second = new Vivus('square', {type:
"oneByOne", start : "manual", selfDestroy :
true});
</code>
```

16. Use scenario-sync

Our scenario so far has been delays and durations defined within the constraints of an overall time frame. If we wanted to, we could have even more control over when our animation starts and stops by creating a scenario-sync time of animation. A scenario-sync animation draws each line one at one time, rather.

17. Forget old paths

Now create a new SVG.

```
<code>
<svg id="curve">
  <rect width="200" height="200" style=
  "fill:rgba(255,255,255,0); stroke-width:10;
  stroke:rgb(40,40,40)"/>
  <polygon points="0,0 20,200"
  style="stroke:rgb(40,40,40); stroke-
  width:5;"/>
  <polygon points="200,180 180,0"
  style="stroke:rgb(40,40,40); stroke-
  width:5;"/>
</code>
```



Defining SVG

When you first start coding SVG graphics into HTML, you may find the whole process a little daunting. Defining things in markup is something we, as web developers, try to avoid using unless we really have to. Positioning elements can be fairly difficult too, when you're defining points for an SVG element, you aren't passing pixel values for the screen, rather, you're passing pixel values offset by the location of the parent SVG container. If your container is 40 pixels left by 20 pixels down and you create a path at 10 by 10, it will start drawing at 50 by 30 pixels on your screen. In this regard, SVG is much akin to Canvas.

```
<polygon points="0,20 40,200"
style="stroke:rgb(40,40,40); stroke-
width:5;"/>
```

For the full step code see 'SVG-Step13' in the tutorial files on FileSilo.

For this final demo we have three new attributes that we can use to control our animation - data-delay, data-duration and data-async. Data-delay and data-duration do what they say on the tin, data-async is a little different. If you have a bunch of paths that you want to animate at the same time, rather than one after another, data-async will enable that for you.

```
<code>
...
<svg id="curve">
  <rect width="200" height="200" style=
  "fill:rgba(255,255,255,0); stroke-width:10;
  stroke:rgb(40,40,40)"/>
  <polygon points="0,0 20,200"
  style="stroke:rgb(40,40,40); stroke-
  width:5;" data-delay="0" data-
  duration="10" data-async/>
  <polygon points="200,180 180,0"
  style="stroke:rgb(40,40,40); stroke-
  width:5;" data-duration="10"/>
  <polygon points="0,20 40,200"
  style="stroke:rgb(40,40,40); stroke-
  width:5;" data-delay="20" data-
  duration="20" data-async/>
  <polygon points="200,160 160,0"
  style="stroke:rgb(40,40,40); stroke-
  width:5;" data-duration="20"/>
  <polygon points="0,40 60,200"
  style="stroke:rgb(40,40,40); stroke-
  width:5;" data-delay="20" data-
  duration="20" data-async/>
</code>
```

```
duration="30" data-async/>
<polygon points="200,140 140,0"
style="stroke:rgb(40,40,40); stroke-
width:5;" data-duration="30"/>
...
</code>
```

18. Vivus and fills

In some of our examples, you may have noticed that there is a <rect> shape included and it doesn't behave as one might expect. This was to illustrate one of Vivus caveats. Though Vivus is great at handling animation between two points, it can't animate <circles>, <rects> or other SVG elements such as text. Though this is a shame, it's not the end of the road for us though because pathfinderjs is included. Pathfinder takes your SVG shapes and converts them into paths that it can animate. The effect might not be the same but it's better than halving your content. A more noticeable problem with Vivus is that it can't actually animate any fills, for now, the fill will be shown straight away and then the paths will be drawn around it.

19. Callbacks

Like most good JavaScript libraries, Vivus enables us to fire callbacks after it has completed the action we asked it to do. If we want to pass a callback to a specific animation, we simply pass a function as a third argument to that SVG's animation object, and just like that we have mastery over animated SVGs using Vivus.

```
<code>
var scenario = new Vivus('nightSky',
{type: "scenario"}, function(){
  console.log("The callback has been
  called!");
});
</code>
```


↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Code on-scroll image animations with CSS

inspiration bloomberg.com/graphics/2015-sepp-blatter-fifa

Whether it's for fun or to emphasise something serious, this sinister on-scroll effect can be used to capture the attention of your website viewers and to set the tone. With this in mind, there can be justified use of this effect for user experience design as well as using it from a graphic design perspective.

This animation effect makes use of HTML, CSS and JavaScript. This is important because CSS alone can't be used to detect page-scrolling events, hence these interactions are managed with JavaScript while the default styling settings are defined in regular CSS.

Note that for this Web Workshop we have placed both CSS and JavaScript in separate files so that the effect can

be reused across multiple pages without any code duplication taking place.

The CSS filter property is fairly versatile and allows for many other options in addition to or instead of the greyscale property used in this tutorial.

It would be great to see how this concept is adapted to produce different types of sinister effects, share them with @webesignermag on Twitter.





<comment>
What our
experts think
of the site

Making something look serious

"The sinister effect can be used to emphasise something serious and is ideal for use on webpages promoting a serious topic. In the case of the Bloomberg Business website, it is used to set the tone of a Fifties gangster movie, which have been the stage for of many stories about corruption."

Leon Brown, freelance web developer

Technique Code the image

1. Main page setup

The main page requires the standard HTML, head and body containers to be defined. This also enables us to insert the page components in the following steps in a way that keeps the JavaScript and CSS separate from the page body content.

2. CSS file linking

This HTML markup will link the files that the CSS and JavaScript code are contained in to the page so that the styling and functionality can be shared across multiple pages. It also enables us to keep the code clean by separating content, styling and functionality.

3. Content containers

Insert the sinister header and the main page content inside the page <body> tag. The sinister <header> contains the background photo image, an overlay image used for the zoom and a content container for the additional text content - in this case it's a title.

4. Initiate page styling

Now that the HTML elements are in place, we can move to the styles.css file to start styling the page. We start this file with the default styling of the main page - mainly the page background and the <main> content container, which we want to use to ensure that the content is centred and has readable text.

5. Sinister styling

The header's styling need to stretch across the page and have a visible height. It will also need to hide any overflow from the zooming images we are using. Images and the content container will be posited at the top left of the <head>, which is made possible with the header using relative positioning.

```
$(window).on('scroll', header{
display: block;
position: relative;
height: 100%;
text-align: center;
overflow: hidden;
margin-top: 25%;
}
header h1{
font-size: 6em;
color: #c00;
```

```
text-shadow: 2px 2px #000
}
header img,
header .content{
position: absolute;
top: 0;
left: 0;
margin: 0 auto 0 auto;
width: 100%;}
```

6. Style individual images

There are two images used in our layout - the first being the main photo image and the other being the sinister zoom overlay. We've avoided overcomplicating the HTML with class name and instead are using their position in the <header> container to define positioning styles.

```
$( "#header" ).click(function() {
if (menuOn == false){
$('#menu').animate({"bottom": -100}, 500 );
menuOn = true;
} else {
$('#menu').animate({"bottom": "-100%"}, 500
);
menuOn = false;
}
});
});
</script>
```

7. Initiate listening code

With the styling now complete, we are ready to add the code to trigger changes as the page scrolls. This is achieved by adding JavaScript code that waits for a page-scroll event. We put this inside another listener for the completion of the page loading to avoid an error.

8. Activate image zooming

The sinister effect is primarily made from two animations - the main picture made bigger and the overlay made smaller to focus its inverted transparent circle around the image. We use a query selector to target the images to

EXPERT ADVICE

Adapting the sinister effect Sinister features

Additional features can be added to the effect by adding new JavaScript code within the scroll listener. This tutorial uses the greyscale filter property as an example of changing the image colour, but other options are also available for experimenting.

Calculations

The calculations for the transition are made using percentages so that it works the same in different resolutions. This is important to ensure that the effect doesn't break with high-res screens. You can test this using the zoom-out features of your web browser to simulate a higher resolution.

Limitations

Step 8 shows a condition that stops the overlay resizing once its horizontal position exceeds -10 pixels. This prevents the illusion from being broken by larger resolutions that stops the overlay covering full width, keeping the webpage adaptable and future-proof.

apply sizing and positioning calculations based on the scroll position.

```
/*-- PUT THIS INSIDE THE SCROLL LISTENER
document.querySelector("header img").style.
width = (100+(window.scrollY/20))+"%";
document.querySelector("header img").style.
left = (0-(window.scrollY/50))+"%";
if(-200+(window.scrollY/3) < -10){
document.querySelector("header img:nth-
child(2)").style.width = (500-(window.
scrollY/1.5))+"%";
document.querySelector("header img:nth-
child(2)").style.left = (-200+(window.
scrollY/3))+"%";
}
if(-180+(window.scrollY/3.5) < -20)document.
querySelector("header img:nth-child(2)").
style.top = (-180+(window.scrollY/3.5))+"%";
```

9. More sinister colouring

The effect can be made to look more sinister by making the main photo change from full colour to black and white as the sinister effect takes place. This can be done by using the scroll position to affect the greyscale property. For the full code on this tutorial, make sure that you check our FileSilo site.

“ The effect can be made to look more sinister by making the main photo change from full colour to black and white as the sinister effect takes place ”

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Create animated infographics with Snap.svg

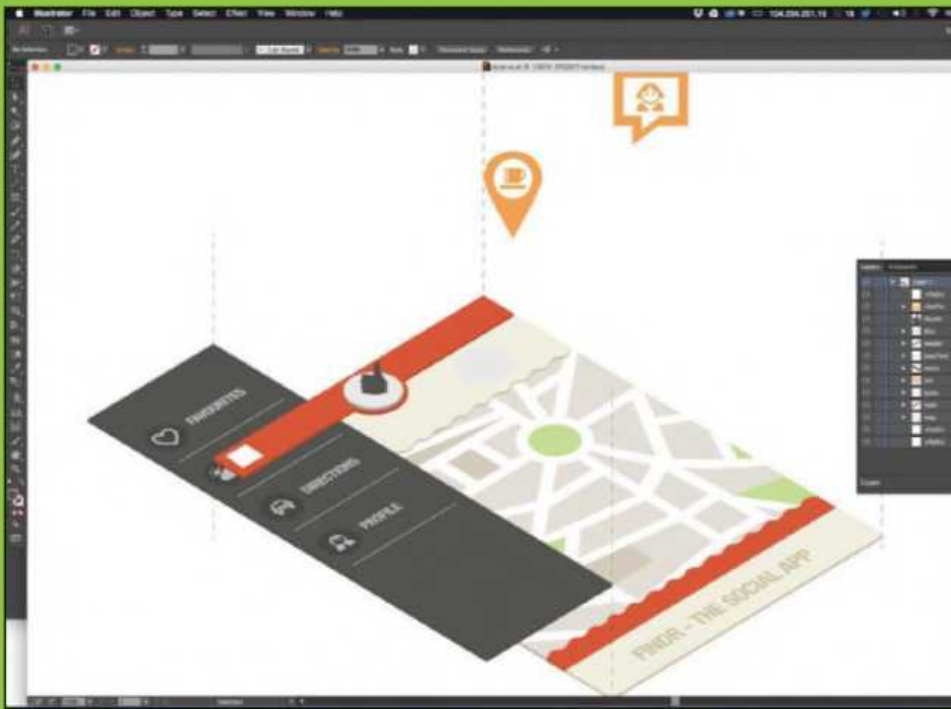
Take a static graphic and add interactivity and animation to create an exploded-view infographic

In the new responsive web world that designers have to inhabit, sizing graphics is of the utmost importance to us as web designers. Getting crisp graphics on both a phone and on a huge cinema display monitor is not just desirable, but essential. To that end most of us have switched to using SVG images for our icons and line art drawings, which are always

crisp on every display. Not only that but the file size is often much smaller than a PNG image. When it comes to creating animated images or interaction with graphically rich content we will need a solution that can help us do this too. Thankfully Snap.svg comes to the rescue because it lets us use crisp SVG graphics but also adds the polish that clients have come to love for their shiny web apps.

We are going to use Snap.svg to create one of those exploded-view images that you would find in Haynes' car manuals but we're going to dissect a mobile app. We'll take a regular SVG created in Illustrator and add animations, then make the buttons on the interface work. These will drop pins onto a map and slide an off-canvas menu onto the screen. This provides a great introduction to making your own graphically rich apps using Snap.svg.





<Left>

• In Illustrator you can see all of the graphics named in the layers panel, so make sure that you open up the layer to see the names. In the background of the SVG element, a background image has been added through CSS.

<Top left>

• In the background of the SVG element, a background image has been added through CSS.

<Top right>

• After loading the graphic it is displayed over the top of the background image, but still needs a little work.

1. Start the project

To start off the project, take the tutorial files from FileSilo and open the Start folder in a code editor such as Brackets. In the head section of the index.html page add a link as shown to the Snap.svg library. You can also download the library from snapsvg.io for reference.

```
<script src="js/snap.svg-min.js"></script>
```

2. Style the SVG element

Just after the Snap code library has been imported, the CSS styling for the SVG element is set. This is displayed, centred on the page and a background image is placed in there so that all the vector lines and shapes have a background to stand out against.

```
<style>
svg{
display: block;
margin: 0 auto;
background-image: url(bg.jpg);
}
</style>
```

3. Start up Snap.svg

After the CSS styling in the head section, script tags are added with an onload function that detects if the page has finished loading.

Once the page has been loaded up, two variables will be declared. One of these variables will hold a timer that waits a fraction of a second before starting the animation and another variable holds the Snap SVG file.

```
<script>
window.onload = function () {
var timer;
var s = Snap(1280, 800);
};
</script>
```

4. Load an SVG file

Before the closing bracket of the onload function in the previous step, add the code shown here. This tells snap to load the scene.svg file. Notice the comment that states the 'tutorial code goes here' - this is where the rest of the tutorial code will go. Because once the SVG is loaded, the code between the curly brackets is called.

```
Snap.load("scene.svg", function(f) {
//TUTORIAL CODE GOES HERE
});
```

5. Reference individual graphics

Once the scene has finished loading up, we will need to do an important part of the tutorial - we need to reference individual graphic elements so that they can be animated or interacted with on the page. Here the variables will reference unique IDs, and these IDs are the ones that you'd find on any regular HTML tag. They will be stored in a variable to be used later on.

```
var menu = f.select("#menu"),

pin = f.select("#pin"),
chatPin = f.select("#chatPin"),
thumb = f.select("#thumb"),
```

Coding Snap.svg

Snap.svg uses JavaScript to manipulate the still SVG images that you import into it. The code adds animation and interactivity to the graphics for a rich user experience.

```
disc = f.select("#disc"),
```

6. More references

Notice that each line of the code shown below has the variable that is then followed by the letter f. If you take a look at Step 4's code, you will notice that the f included there appears inside the brackets of the function, and all of this code is in that function. The letter f simply stands for file and it is a reference to the SVG file that we had set to load from before.

```
header = f.select("#header"),
icons = f.select("#icons"),
base = f.select("#base"),
baseText = f.select("#baseText"),
```

7. Last graphic references

Here the last references are stored in variables and you may be wondering where exactly these names came from or how they got in the SVG file. When graphics are grouped together in Illustrator, they can be named in the layer panel, these names are applied as IDs when exported into SVG.

```
map = f.select("#map"),
```

jQuery & JavaScript



<Top left>

- A little tidy up is required as some graphics aren't needed until animation is applied to other elements

<Top right>

- The first graphic elements are animated into position, showing an exploded view of an app

<Right>

- Interactivity is added to the other icon but there needs to be a way to reset this so the icon goes away again



Animating scale

If you want to animate the scale of an image, then SVG is the perfect graphic to scale as it is a vector and it can be blown up in size without any loss of quality.

```
people = f.select("#peopleBtn"),
chat = f.select("#chatBtn"),
burger = f.select("#burgerBtn");
```

8. Display the SVG on the page

The next line of code is quite an important line because this line displays the graphic on the screen. If you remember that `s` is the Snap reference, then append means to add to that, so we use the reference of `f` which is the SVG file. Save the page now and hit the 'live preview' in Brackets or view the page from a server to see the graphic.

```
s.append(f);
```

9. Clean up the graphic

At present there are some pins floating in space and a menu that is obstructing the way of the main graphic. These will be important to us later but for now let's make them invisible.

Place the code here, just before the line we added in Step 8 as this effect should be applied before the image is displayed.

```
menu.attr({ opacity:0 });
pin.attr({ opacity:0 });
chatPin.attr({ opacity:0 });
```

10. Hide 'hit areas' of buttons

You can refresh the browser now and see the updates to the graphic from the previous step. There are some hit areas for buttons to hide the button icons, so those will need to be made invisible as well. Add this code that we've provided below immediately after the code from the last step.

```
people.attr({ opacity:0 });
chat.attr({ opacity:0 });
burger.attr({ opacity:0 });
```

11. Leave a pause

Once the graphic is on the screen, the animation is ready to be applied but it should wait a little before starting – this is so that the user can see the graphic, then see it as it moves into position. Add the code here after the `s.append(f)` from Step 8. This code will set a timer to wait for 300 milliseconds.

```
timer = setTimeout(designIn, 300);
function designIn() {
  clearTimeout(timer);
```

12. Animate the graphics

Add the code to finish the function from the previous step. These graphics are animated with a slight bounce to get them into the position we want. The `t` inside the transform means that they are transformed relatively from their current position. The timing of each is placed after this. Then just hit Save and view it in the browser to see the animation.

```
thumb.animate( {transform: "t0,-160"},1500,
mina.backout );
```

```
disc.animate({transform: "t0,-150"},1100,
mina.backout);
header.animate({transform: "t0,-130"},850,
mina.backout);
icons.animate({transform: "t0,-100"},550,
mina.backout);
}
```

13. More animation

The graphics that are at the bottom of the app will also need to be animated, and so does the map in the background. Add the following lines of code before the closing bracket that we showed you in the previous step. Now just save the file and view it in your browser to see all the elements animate into position on the screen.

```
baseText.animate({transform: "t0,-130"},850,
mina.backout);
base.animate({transform: "t0,-100"},550,
mina.backout);
map.animate({transform: "t0,-70"},450, mina.
backout);
```

14. Move the thumb icon

As it stands, the animation stops moving but it would be good if we could make the thumb continuously bob about in the graphic. Go back to the code added in Step 12 and amend the 'thumb' code as shown. This will call another function `thumbUp` when it has finished.

```
thumb.animate( {transform: "t0,-160"},1600,
mina.backout, function(){thumbUp()} );
```

15. Move the thumb up

The `thumbUp` function is added here and it needs to go

just below the closing bracket of the designIn function. This 'thumbUp' function is called after the thumb has moved into position and it moves it up a little higher with easing so that it slowly builds up speed and then subsequently slows down again.

```
function thumbUp()
{

thumb.animate( {transform: "t0,-210"},800,
mina.easeinout, function() {thumbDown()} );

}
```

16. Move back again

Once the thumb has moved up, it calls the function thumbDown so now we will add that function below. Again this moves the thumb down and calls the thumbUp function that keeps the thumb bobbing up and down in a loop. Now just save this and view it in the browser, as we have done previously, to see the animation in action.

```
function thumbDown()
{

thumb.animate( {transform: "t0,-160"},800,
mina.easeinout, function(){thumbUp()} );

}
```

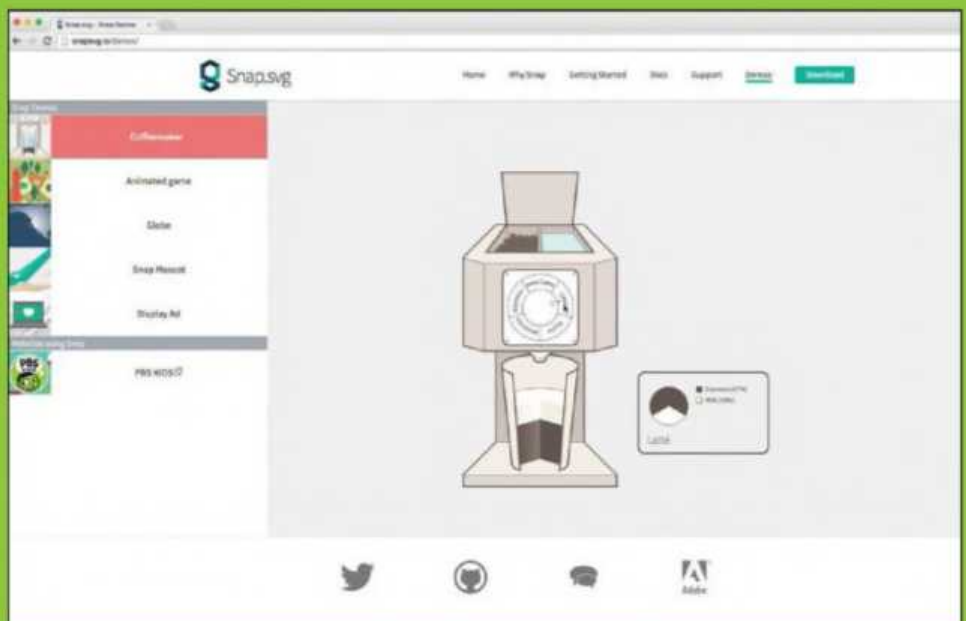
17. Add interactivity

It's time to make the graphic interactive now. Adding the next set of code will make the icon of the person a button. Clicking on this button will make the pin animate into the graphic and bounce onto the map. The pin was made invisible in Step 9. Save your code and view it in a browser to test the click.

```
popcorn.cue( 17.2, people.click(function ()
{
pin.animate({opacity:1, transform:
"t0,280"},900, mina.backout);
});
```

18. Click the chat icon

Similar to the previous step, if the user clicks on the chat icon, the chat pin should drop onto the map and show the location of the chat. The code adds this on-click animation, and once this is done you can save the page and view it in a browser to test that the click is working.



Other ways to use Snap.svg

Snap.svg is great at manipulating premade SVG images but it is also great at creating content from scratch, perfect for making basic geometric shapes like circles or rectangles. There is a great tutorial on this at the Snap.svg site snapsvg.io, but there are also some other great examples at websites like svg.dabbles.info that help to fill in some blanks in working with Snap.svg. Of course the perfect match for graphics is to create some graphics in your favourite vector illustration program, then add to it by creating dynamic masks with shapes in Snap.svg. An excellent example of this is the Coffee Maker demo that comes when you download the source code of Snap.svg, but bare in mind that this is a very advanced application!

```
chat.click(function () {
chatPin.animate({opacity:1, transform:
"t0,280"},900, mina.backout);
});
```

19. Reset the pins

When the user clicks either of the icons and the pins drop into position, they will stay there. It's possible to only show one at a time by making the first pin animate back off when the other icon is pressed. Just add this line of code to the click function of Step 17 to do exactly that.

```
chatPin.animate({opacity:0, transform:
"t0,0"},900, mina.backin);
```

20. Adjust the second pin

As in the previous step, the code here needs adding to the click function in Step 19. This will make the first pin disappear when the other icon has been clicked. This is a simple solution for what we wanted to achieve so save

this, then just test this in the browser and only one pin will display at a time.

```
pin.animate({opacity:0, transform:
"t0,0"},900, mina.backin);
```

21. Bring in the menu

Now let's make the burger icon a button that can be clicked and this will bring the off-canvas menu into the app. Add this code below, which will animate this onto the screen. Save this and test it.

```
burger.click(function () {
menu.animate({opacity:1, transform:
"t150,-70"},900, mina.backout);
});
```

22. Remove the menu

To get rid of the menu, it is as easy as letting the user click on the actual menu itself. Adding the following code will animate it back to the original position, while also taking the opacity down and making the menu invisible. Save this and view again in the browser to see and test all the functionality.

```
menu.click(function () {
menu.animate({opacity:0, transform:
"t0,0"},900, mina.backin);
});
```

With its charting heritage, Highmaps content is also treated as a series of datapoints and the options for these are configured here

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Prototype apps with Framer.js and Framer Studio

Make your app's wireframes come alive with animations and interactions using Framer.js **tools | tech | trends** Framer.js, Framer Studio, Photoshop CC, Mac OS X



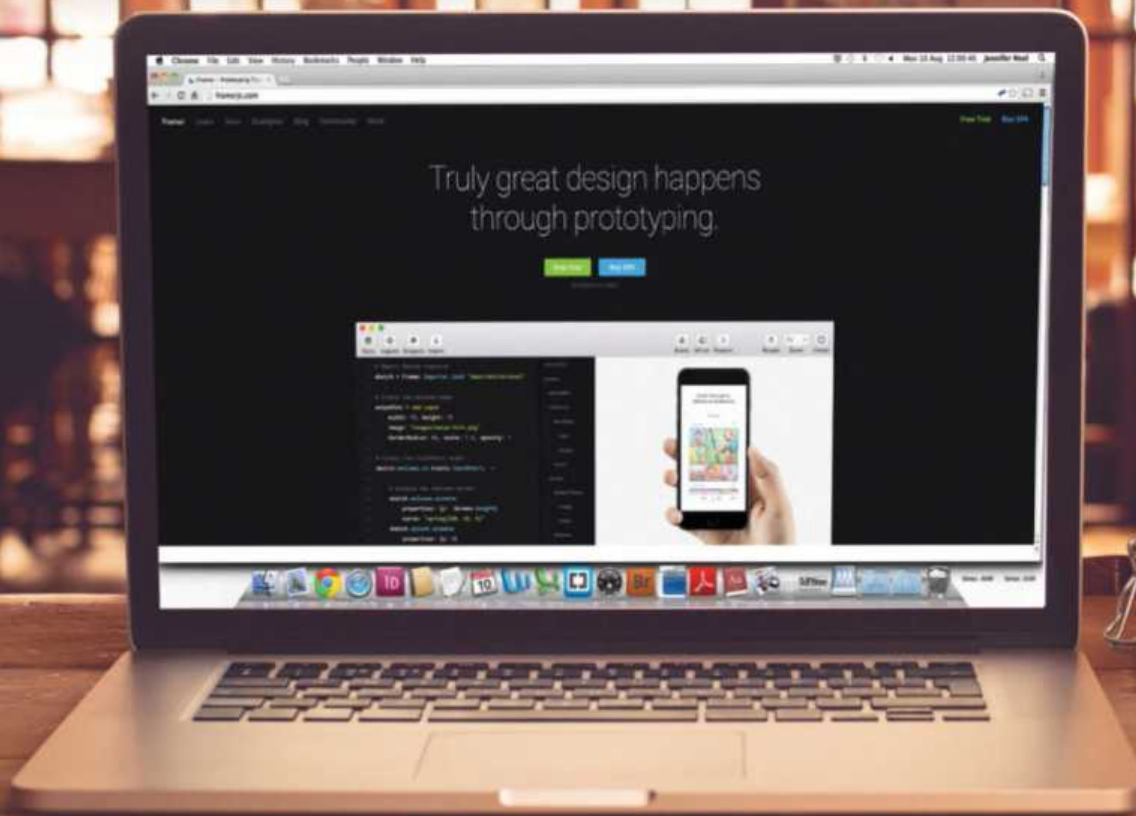
App development is hard, no two ways about it. Wireframing, user experience, user stories, colour palettes – the whole app is designed before a line of code is ever touched! But

sometimes, decisions made in the design process don't translate well to development. Take animations,

for example, what a designer might imagine is supereffective at communicating an idea to somebody might actually be a terrible design decision – and this might not be discovered until a point in the app development cycle that could make it quite difficult to change.

Building a bare-bones app to test out ideas and interactions is still really time-consuming, you need to

have a development environment set up, you need to have a device, its simulator or its SDKs. But what if you've never coded before? This is where Framer.js comes in. Framer.js and Framer Studio are pieces of software that use as little code as possible to make as much awesome as you can think of. For this tutorial, we're going to import an app design from Photoshop bind events to certain layers and animate interactions.



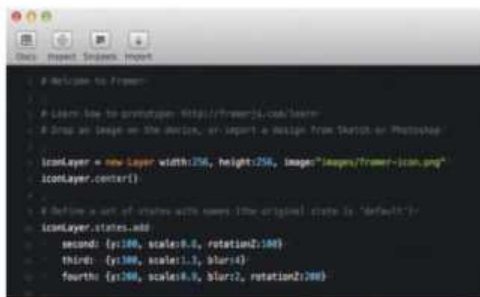
1. Get Framer.js

Framer Studio is the software that we'll be using to put our app prototypes together. It costs \$79.99 (£51), but there is a 14-day free trial that gives us access to the full range of tools. Just enter your name and email over at framerjs.com and a link will present itself.



2. First impressions

When we first run Framer Studio, we're presented with a bare-bones app. In the image below, our CoffeeScript code view that we'll be using to program interactions and animations can be seen.

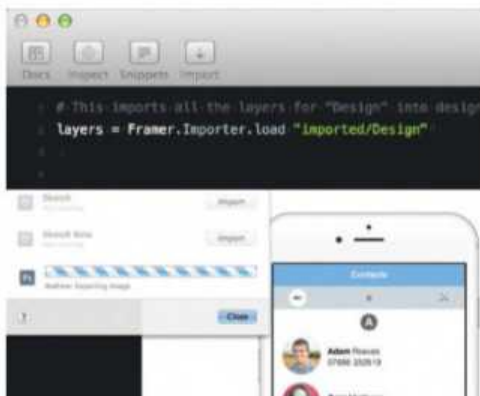


3. Create a project space

Before we get started, we need to save a project to our disk so we can conveniently access and import assets as we need them. We'll use the default project that we've been given as a starting point. Go to File>Save and pick a place to save our Framer project and resources.

4. Import a PSD

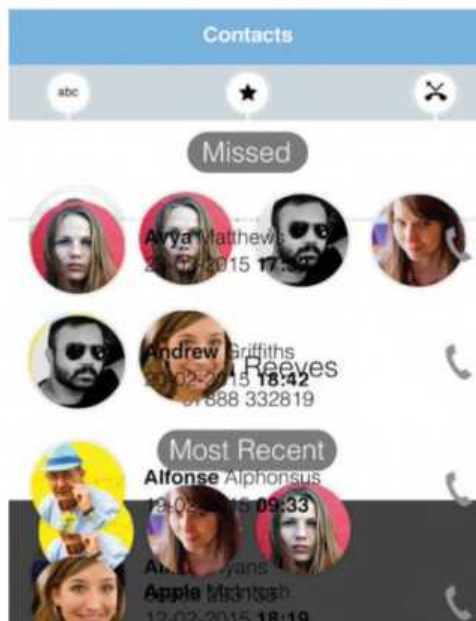
Framer can import both Sketch projects and PSD files. For this tutorial we're going to import and add interactions and animations to a PSD to prototype a simple contacts app.



You can grab the design.psd file from this projects assets on FileSilo. Open the PSD in Photoshop CC and click the Import button in the top-right corner of Framer Studio chrome.

5. Look at the PSD

When we open our PSD file in Photoshop, we see that it's a bit of a mess. This is because any layers that we hide won't be imported into Framer Studio. In order to get everything we need for our prototype into our Framer project we need it to be visible, then we use some simple code to hide the appropriate layers.



6. Complete the import

Just like in our PSD file, our Framer project is really messy, with layer upon layer of... layers. At the top of our code view, you'll notice that a line has been added for us.

```
<code>
layers = Framer.Importer.load "imported/
Design" </code>
```

This is a JavaScript object that we can use to access all of our layers. How do we know what layers to access? Simple, the group names and layer names that we've used in the PSD are the names that are given to our Framer layers.

7. Create variables

To help us tidy up and access layers easily. We're going to create some variables. We can use these to affect properties, like opacity and bind events:

```
<code>
Framer.Defaults.Animation = {
  curve: "linear",
  time : 0
}
</code>
```

The outline property

The outline shorthand property sets all the outline properties in one declaration. The outline-color, outline-style and outline-width can all be set.

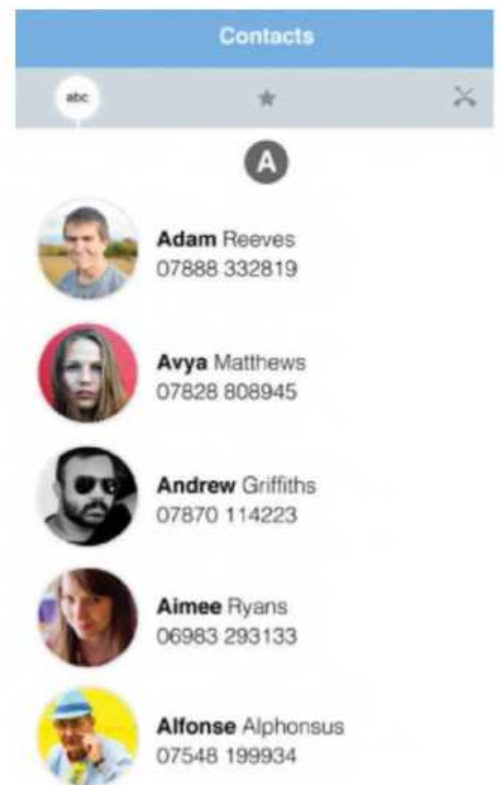
```
tab = layers.Tab
icons = layers.Icons
alpha = layers.alphabetical
favourites = layers.favourites
missed = layers.missed
```

8. Hide the layers

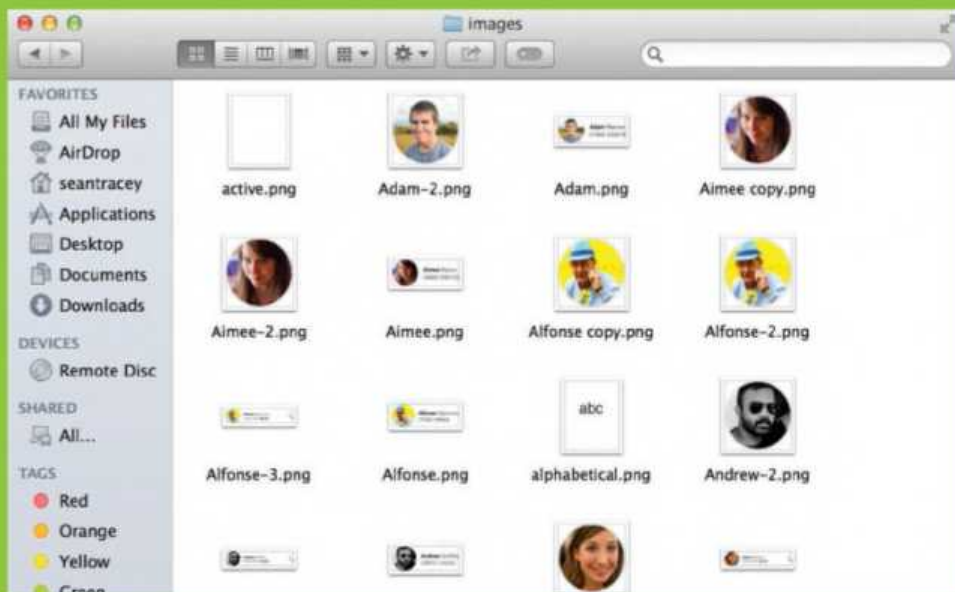
Now that we have variables for our main layer groups, we can start hiding the ones we don't want to see when we start our app prototype up. Just after our variable names, add the following code:

```
<code>
...
favourites.opacity = 0
miss.opacity = 0
favs.opacity = 0
missed.opacity = 0
adamDetail.y = 1140
...
</code>
```

Our app preview window should refresh straight away leaving us only with the alphabetical contacts view.



jQuery & JavaScript



Layers and layers

Layers in Framer are designed to behave like layers in Photoshop and Sketch. When we export our PSD to Framer, what's actually happening is Framer Studio is working through the tree of our Photoshop document and creating objects that represents each individual layer as an asset. This is then saved as a PNG and then stored in the /imported/Design/images folder within our project folder. Framer ignores everything that's hidden in the Photoshop layers, which is why we leave everything visible as we export it, if a layer is transparent it will be exported at that transparency, tweaking the opacity value of that layer can't make it more opaque, only more transparent.

9. Add states

To show and animate layers in our project, we can add states. These are simple objects that will describe what properties our layers should have when that state is active. Think of them a little bit like CSS classes that we can toggle on and off.

```
31 alpha.states.add
32   - default : {opacity : 1}
33   - second : {opacity : 0}
34
35 favourites.states.add
36   - default : {opacity : 0}
37   - second : {opacity : 1}
38
39 missed.states.add
40   - default : {opacity : 0}
41   - second : {opacity : 1}
42
43 abc.states.add
44   - default : {opacity : 1}
45   - second : {opacity : 0}
46
47 favs.states.add
48   - default : {opacity : 0}
49   - second : {opacity : 1}
50
51 miss.states.add
52   - default : {opacity : 0}
53   - second : {opacity : 1}
```

10. Add events

Now that our layers have states, we can write code to toggle certain states when different layers are clicked or touched. We're going to bind some click events to the tabs at the top of our app, and have them show the view that they correspond to. Go ahead and tap one of the tab buttons, the view should now switch out.

```
alpha.on Events.Click, ->
  - alpha.states.switchInstant('default')
  - favourites.states.switchInstant("default")
  - missed.states.switchInstant("default")
  - abc.states.switchInstant('default')
  - favs.states.switchInstant('default')
  - miss.states.switchInstant("default")
```

<code>

```
...
favourites.on Events.Click,->
favourites.states.switchInstant('second')
alpha.states.switchInstant("second")
missed.states.switchInstant("default")
abc.states.switchInstant('second')
favs.states.switchInstant("second")
miss.states.switchInstant("default")
missed.on Events.Click, ->
alpha.states.switchInstant('second')
favourites.states.switchInstant('default')
missed.states.switchInstant('second')
```

```
abc.states.switchInstant('second')
favs.states.switchInstant('default')
miss.states.switchInstant("second")
</code>
```

11. Understand the events

Even if you're a seasoned JS developer, the syntax in the last step may have looked a little strange. That's because it's CoffeeScript, a language that compiles to JavaScript. All we've done in event handler is set the appropriate states for the tabs buttons and then shown the view corresponding to that tab button whilst hiding all of the others.



12. Add Adam

In addition to importing assets from a PSD or Sketch file, we can also drag and drop files into our project to use them too. In the resources folder of the project on FileSilo, download adam.png and then drag and drop that file into our Framer project. The image will appear in our project and a new line of code will appear at the end of our project.

```
<code>
imageLayer1 = new Layer
x:0, y:0, width:147, height:147,
image:"images/adam.png"
//Change this to...
adamPicture = new Layer
x:38, y:88, width:147, height:147,
image:"images/adam.png", superLayer : abc
</code>
```

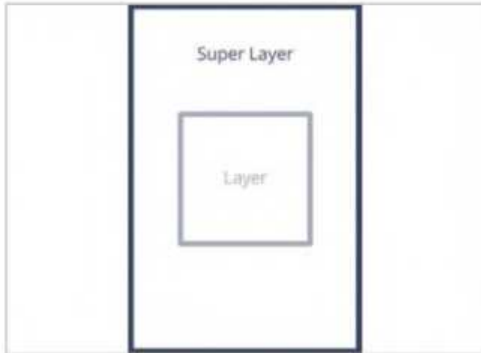
13. Is it a bird, is it a plane? No, it's a superLayer!

After dropping the Adam file into our project, change the code to alter the images position with the x and y properties and add the superLayer property. Our

Animations

When animating in Framer, we have a pretty good set of prebuilt animations for moving stuff around on our prototype. The different types of curve (easing) are linear, bezier-curve, spring-rk4, spring-dho and spring. Each has its own characteristics and tweakable properties.

Framer layers have a hierarchy, using the `superLayer` property, we've told Framer that our `adamPicture` layer should exist with the `abc` layer so that its parent and all movements and actions are relative to `abc`.



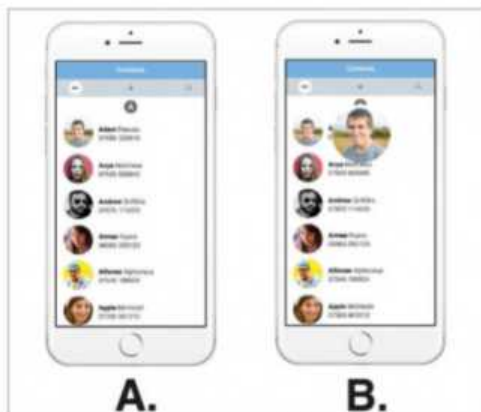
14. Animate Adam

So why, have we added Adam as a separate layer when he's already present in the `abc` layer? Well, because it makes it easier for us to animate him! Just as we did with our tab toggles we're going to add an extra state to our `adamPicture` object as well as create some animation options.

```
<code>
...
adamPicture.states.add
```

```
adamPicture.states.animationOptions =
  curve: "spring(100,20,5)"
```

```
second: {x : 305, y:100, scale:2}
...
</code>
```



15. Animation options

Almost any property or layer can be animated in Framer Studio. By passing animation options to each layer, we can customise how that element will animate, over any length of time, in response to any action to we trigger it with. In Step 14, we passed through options that would ease our layer to and from points A to B in its animation.

16. Trigger the animation

To trigger our animation we're going to bind an event to our Adam picture exactly the same way we did for our tabs:

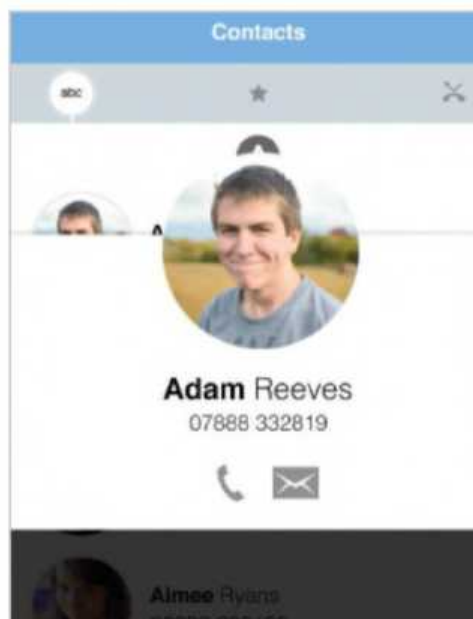
```
...
adamPicture.on Events.Click, ->
adamPicture.bringToFront()
adamPicture.states.next()
...
</code>
```

Rather than setting the states manually, we're using the `next()` function to cycle through the available states. Once it comes to the end of available states, our layer will animate back to the first state in our queue. If you click on the picture of Adam, it should now centre and scale up.

17. Transition details

Adam's face now animates, but not to much effect - he just kind of... sits there at present. In our PSD, we have a detail layer so that as Adam animates to and from his desired spot, we'll have the details layer slide up too.

```
<code>
...
#Add
adamDetail.states.add
second: {y : 140}
adamDetail.states.animationOptions =
  curve: "spring(100,20,5)"
#Edit
adamPicture.on Events.Click, ->
adamPicture.bringToFront()
adamPicture.states.next()
adamDetail.states.next()
</code>
```



18. Show off - Present mode

That's it! We now have a fully functioning, interactive app prototype. So, how do we show this off to people? There are a couple of ways. The most immediate way to demonstrate the prototype is the Present button at the top right of the Framer Studio chrome. Clicking this will launch a fullscreen view presentation mode showing off our app in the current device.



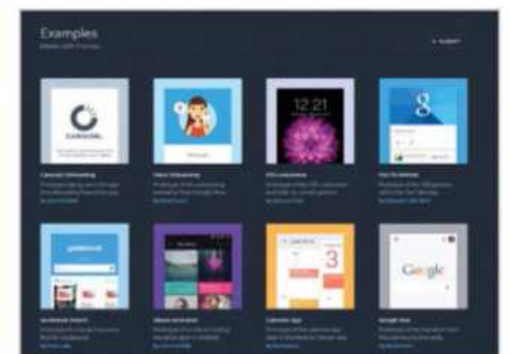
19. Show off - Mirror mode

Another way to demonstrate our app is with the Mirror mode. This will fire up a simple server that will let us visit and interact with our prototype in any Webkit-based (Chrome and Safari for example) browser.



20. Conclusion

That's it. We now have the skills to import a Photoshop-created design, add animation and touch/click events to it for presenting. With all of this, your next project is sure to impress. Check out the examples on the Framer site (framerjs.com/examples) for more inspiration.



🌱 We now have the skills to import a Photoshop design 🌱

Development

112 20 fresh frameworks
Find out the best frameworks

118 Master advanced Angular
Make your Angular code better

128 Construct an image gallery with
CSS and AngularJS
Combine traditions

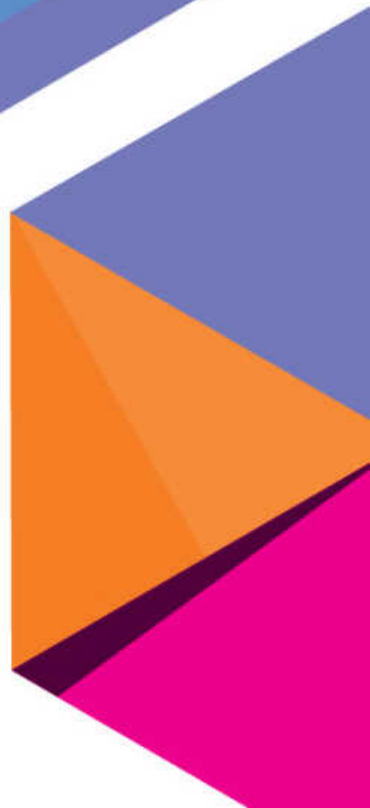
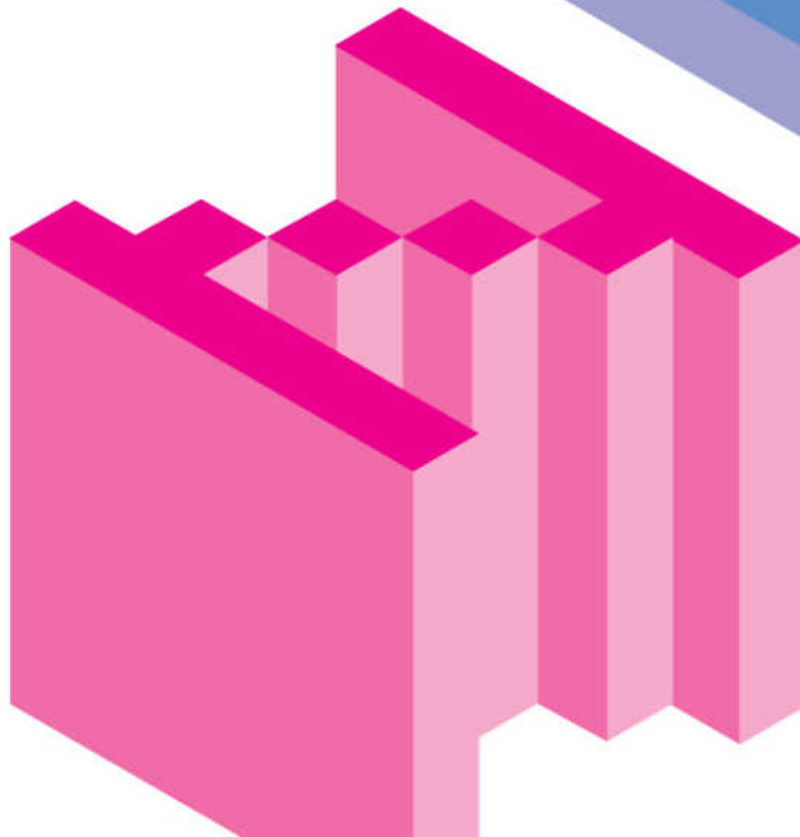
132 Code validation into forms with
ngMessages
Create a new form

138 Create an interactive 360° scene in
WebGL
Make explorable environments

142 Build your own API with the Hapi.js
framework
Develop feature-rich APIs

148 Manage your project's
dependencies with jspm
Master frictionless package
management

☞ It tries to solve the problem of fast-moving visuals with an unorthodox approach. Websites are composed of one or more surfaces, which are displayed by the rendering engine ☞



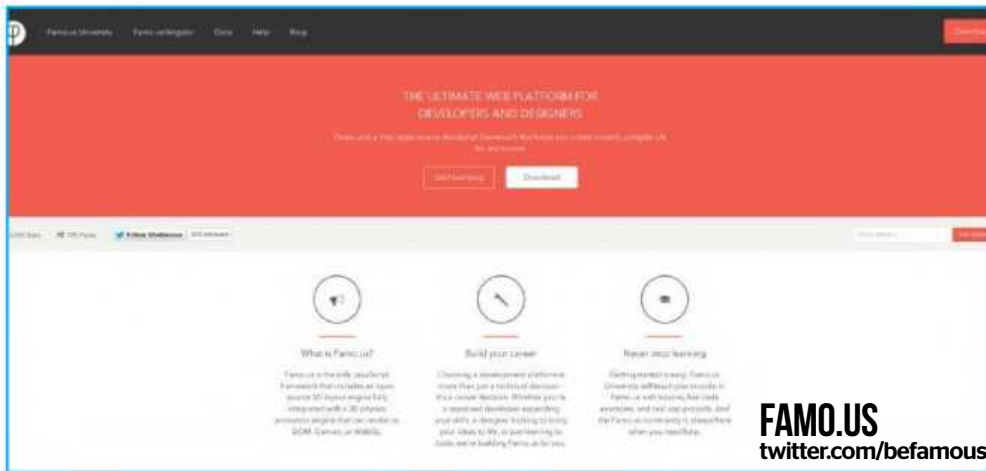


↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

20 FRESH FRAMEWORKS

Frameworks are an integral part of the design and development process. Here we unveil a selection that will get your designs ready for now and the future





FAMO.US

CREATING FAST-MOVING VISUALS IS DIFFICULT DUE TO THE CONSIDERABLE EFFORT REQUIRED FOR DOM TRANSFORMS

Famo.us tries to solve the problem of fast-moving visuals with an unorthodox approach. Websites are composed of one or more surfaces, which are displayed by the rendering engine. Surfaces are handled via an intermediary element known as context. The most basic version of a Famo.us-based website is shown in the snippet:

```
var Engine = require('famous/core/Engine');
var Surface = require('famous/core/Surface');
var mainContext = Engine.createContext();
var firstSurface = new Surface({ content:
"<h3>Hi!</h3><p>I'm a surface!<br>I live
inside a context.</p><p>You can add
<b>HTML</b> content to me and style me with
<b>CSS!</b></p>", size: [200, 200],
properties: {
001 var Engine =
require('famous/core/Engine');
backgroundColor: 'rgb(240, 238, 233)',
textAlign: 'center',
padding: '5px',
border: '2px solid rgb(210, 208, 203)',
marginTop: '50px',
marginLeft: '50px'
}
});
mainContext.add(firstSurface);
```

Famo.us differs from technologies such as WebGL in that the individual surfaces are made up of rendered HTML. This makes the framework ideally suited for all kinds of bouncy content, but creating 3D games is better handled with other technologies for example. Surfaces can even be arranged on top of one another via a group of layout managers such as HeaderFooterLayout:

```
var mainContext = Engine.createContext();
var layout = new HeaderFooterLayout();
layout.header.add(new Surface({
. . .
}));
layout.content.add(new Surface({
. . .
}));
layout.footer.add(new Surface({
. . .
}));
mainContext.add(layout);
```

Keyframe-based animations are a classic use case for the Famo.us framework. Adding motion to a surface involves the creation of a StateModifier object. It is then provided with a group of widgets and one or more curves that will work on describing the beginning and end points of the desired movement:

```
var stateModifier = new StateModifier();
mainContext.add(stateModifier).
add(surface);
stateModifier.setTransform(
Transform.translate(100, 300, 0),
{ duration : 1000, curve: 'easeInOut' }
);
```

Famo.us/Angular permits you to connect to data stored in an AngularJS backend. This simplifies the creation of parametrised animations: a field in a model can be used to determine animation speeds, background colours and similar effects.

FULL HARDWARE ACCELERATION

Apple's iPhone introduced the concept of making the GPU work on animations. Famo.us renders via hardware accelerated CSS transforms, which is very efficient and yields blazingly fast updates whilst consuming minimal power.

ALTERNATIVE HTML FRAMEWORKS

UIKIT

Most GUI frameworks abhor change. This one tries to embrace it

getuikit.com

GOOD FOR: Creating responsive user interfaces that can be themed easily

UIKit is a jQuery-UI-like user-interface framework based on the LESS CSS interpreter. Its use permits the reuse of CSS elements, thereby creating an extraordinarily simple code base that is easy to adjust to your needs.

Sadly, most developers will never need to take a look at the underlying code. UIKit is shipped with a web-based customiser, which permits you to fine-tune most aspects of the rendering engine without touching the actual markup files.

MONTAGEJS

Data binding is so 2010. Montage harnesses the power of the FRB!

montagestudio.com/montagejs

GOOD FOR: Building high-performance single-page apps with advanced data binding

MontageJS is based on various object-oriented design patterns. This means that most components can be expanded easily, thereby aiding the creation of complex applications. A visual designer assists you in getting user-interface markup just right without long and tedious code editing.

Functional Reactive Bindings free you from the obligation of creating glue code. They differ from normal two-way bindings as they permit you to create more complex relationships automatically.

GROUNDWORK CSS

Making typographically impressive user interfaces with minimal extra effort

groundworkcss.github.io

GOOD FOR: Making scalable web apps and responsive text

Developers familiar with Sass will immediately recognise the powerful selectors that simplify the acquiring of pointers to DOM elements. The product ships with a large selection of predefined classes, which create impressively-designed textual elements.

For example, <a> tags with a href attribute pointing to YouTube are automatically expanded with a YouTube icon, providing the user with a visual cue about the contents of said element:

```
<a href="http://youtube.com">Youtube
Link</a>
```

ALTERNATIVE CSS FRAMEWORKS

YAML

Constructing forms and grids can be tedious. This framework helps you out

www.yaml.org

GOOD FOR: Building complex grids without diving into the specifics of layout systems

Time has been unkind to YAML: the framework has been on the market since 2005, but never managed to get into the limelight. Continuous development has created a special-interest product dedicated to the creation of forms, columns and grids.

When working with YAML, start by embedding the CSS files specified in the documentation: due to effective minification, your application's file size increases by about ten kilobytes. In the next step, the actual markup must be added. However, by default, YAML restricts itself to positioning the elements onscreen.

The process of creating the actual form elements involves the adding of `<div>` tags, which are to be parametrised with the class of the element that they are to represent:

```
<div class="ym-column">
<div class="ym-col1">
<div class="ym-cbox">
...
</div>
</div>
<div class="ym-col3">
<div class="ym-cbox">
...
</div>
</div>
</div>
```

TOAST

Grid frameworks can be behemoths. Dan Eden provides a slim alternative [daneden.github.io/Toast](https://github.com/daneden/Toast)

GOOD FOR: Making grids that can act as containers for various UI elements

Grids tend to have an unholy fascination for component vendors, and developers peddle for complex grid components.

But Dan Eden's product goes against the mainstream in that he purposely limits the number of options that is given to developers, in order to create a widget which is very simple to use. Deploying it is as easy as 1-2-3: just start by inserting the file, adding the skeleton tag and then proceed to filling it up.



GUMBY

MANY FRAMEWORKS ARE TIED TO A CUSTOM GUI STACK. THIS FRAMEWORK AUGMENTS WIDGET SETS WITH FLEXIBLE TOGGLES

Getting started with Gumby is easy, all you have to do is add the relevant CSS and JavaScript files to your project. In the next step, controls should be arranged inside the flexible grid component:

```
<div class="row">
<div class="twelve columns">
<p>12 columns</p>
</div>
</div>
<div class="row">
<div class="eleven columns">
<p>11 columns</p>
</div>
<div class="one columns">
<p>1</p>
</div>
</div>
```

GO ADVANCED

Even though Gumby contains a set of Metro-style widgets, its main strength is grouped up in a module commonly known as Components. Elements can be hidden and shown, and stylesheets are toggled automatically as and when they're needed. CSS can be reused via a feature called mixins. For example, responses can be set up in order to modify the structure of the user interface automatically:

```
@include respond(large-screens) {
/* Your font is so big, it's absurd! */
h1,h2,h3,h4,h5,h6 { @include font-size($absurd); }
}
```

Designers will be happy to hear that the Gumby team provides Photoshop templates for most of its user-interface components. They will help to simplify the creation of mockups during the design stage.

START YOUR ENGINES

Reducing the size of the framework makes your web app faster. Gumby provides a tool which strips out unneeded parts of the product i to reduce download volume.



SELECT COMPONENTS

The first part of the tool permits you to select uninteresting components. Simply remove the checkbox in front of any feature that your application will not use.



CONFIGURE ADVANCED OPTIONS

Below that, a selection of text boxes and other widgets permit you to set reasonable defaults for most components. Use this to make your markup simpler.



DOWNLOAD AND DEPLOY

Finally, click the big blue download button in order to obtain your custom build of Gumby. Please make sure to test it thoroughly, as the builder tool is still in beta.





FOUNDATION

CAN ZURB BACK UP ITS CLAIM OF BEING THE MOST ADVANCED RESPONSIVE FRONT-END FRAMEWORK IN THE WORLD?

Developers who download and unpack the basic Foundation distribution are treated to a complete archive containing a turn-key website including both logic and stylesheets. ZURB's documentation suggests that the individual sites are to be based on the grid system of the framework. A module called Interchange lets your website select different source files as the size of the screen changes – the following example uses three different pieces of content for devices with small, medium and large displays:

```
<div data-interchange=["../examples/
interchange/default.html, (small)], [../
examples/interchange/medium.html, (medium)],
[../examples/interchange/large.html,
```

```
(large)]">
<div data-alert class="alert-box secondary
radius">
This is the default content.
<a href="#" class="close">&times;</a>
</div>
</div>
```

NAVIGATIONAL AIDS

Foundation provides a batch of classes which simplify the creation of navigational elements. Off-canvas menus can be added to your website with a group of predefined tags – you are freed from the obligation of coding everything by hand in CSS. iOS-styled navigation bars are realised via the icon-bar class.

Should you work on a multimedia application, you will be delighted to discover that ZURB provides an attractive and responsive lightbox component for this purpose. Users can click a single thumbnail in order to open the fullscreen gallery viewer. Captions and other navigational aids are rendered automatically if enabled.

positioning. Finally, a set of dedicated tags display PHP, CSS and HTML code with syntax highlighting.



99LIME.COM/ELEMENTS
twitter.com/htmlkickstart

SKEL.JS

RESPONSIVE LAYOUT SYSTEMS TIE THEIR USERS TO A SPECIFIC GUI STACK. SKEL IS DIFFERENT

Adding Skel to your website immediately changes the behaviour of your client's browsers. This is because a group of stylesheets are added, thereby overriding some of the more annoying per-tag formatting specified by browser vendors.

iFrame-based videos should be embedded inside a flex-video class. It ensures that the clip's scale remains correct, independent of the screen size and display settings of the client's device:

```
<div class="flex-video">
<iframe width="420" height="315" src="//
www.youtube.com/embed/<VID>" frameborder="0"
allowfullscreen></iframe>
</div>
```

GUI BONANZA

Our screenshot shows a part of an overview known as the Foundation Kitchen Sink. It contains one instance of every control found in the exhaustive GUI stack of the framework. Feel free to select from alerts, buttons, drop-down lists, tables and loads of other widgets which might help your application.

COMPATIBILITY

Developers working on international applications will be delighted to know that the framework comes with support for right-to-left languages such as Arabic.

The current version of Foundation is compatible with all major mobile and desktop browsers. Sadly, its support for Internet Explorer is limited to IE9 and up. Any users who are stuck on older versions of Microsoft's workhorse will have to live with all kinds of weird issues.

SASS INTEGRATION

Typing long lists of boring markup is a time waster. ZURB develops Foundation using the SCSS preprocessor: uncompiled source files are provided for programmers working on Sass-based projects.

HTML KICKSTART

CREATING IMPRESSIVELY-LOOKING WIDGETS IS HARD. 99LIME PROVIDES A SET OF COOL ONES IN A DEDICATED FRAMEWORK

Lists, tables, buttons and styled text blocks require significant formatting. Embedding HTML5 Elements solves most of these problems. For example, creating coloured buttons becomes as easy as adding an attribute or two:

```
<button class="blue"><i class="icon-
star"></i> .blue</button>
```

In addition to that, the framework comes with the well-known icon set from Font Awesome. This means that you can use a few hundred well-known symbols without having to play around with images and

Since users can change the size of the window, the Breakpoint Manager feature provides callbacks, which notify your website when specific sizes are reached:

```
skel.on('+small', function() {
/* Turn on feature for small displays */
});
skel.on('-small', function() {
/* Turn off feature for small displays */
});
```

Skel also provides a simple responsive grid that can be used for arranging controls on the screen:

```
<div class="row">
<div class="2u">Two</div>
<div class="4u">Four</div>
<div class="6u">Six</div>
</div>
```

GETSKEL.COM
github.com/n33/skel



TYPESCRIPT

THE SCALABLE LANGUAGE COMPILES TO JAVASCRIPT ON ANY HOST, BROWSER, AND OPERATING SYSTEM

TypeScript wants to be a better JavaScript. It achieves this goal by resorting to the tired and true approach of **transpiling**. This means that your code is transformed to actual JavaScript after hitting the save button on the editor. During this process, nonstandard bits of code get replaced with glue logic; potential errors can be found via static analysis.

Microsoft's biggest addition involves the availability of formalised class declarations. As an example, look at the sample greeter class:

```
class Greeter {
  greeting: string;
  constructor(message: string) {
    this.greeting = message;
  }
  greet() {
    return "Hello, " + this.greeting;
  }
}
```

Complex object oriented designs can be erected via mixins and interfaces; basic support for generics enables you to create super-reuseable data types, which expand the usability of logic. Very large programs can even be compartmentalised by the use of modules. They permit you to hide and expose logic as needed - the module shown in the snippet exposes but one of its members:

```
module Validation {
  export interface Validator {
    isOK(s: string): boolean;
  }
  var lRegexp = /^[A-Za-z]+$/;
  var nRegexp = /^[0-9]+$/;
  . . .
}
```

TypeScript's relatively rigid syntax improves IDE support. Recent versions of Visual Studio deliver amazingly intelligent suggestions when asked to complete half-typed statements. Accessing code written from third parties is greatly simplified: well-written libraries check the correctness of invocations at compile time.

DECLARE A VARIABLE TYPE

In JavaScript, a field declared with `var` can take any type. TypeScript permits you to specify the type of value stored in a variable at compile time.

```
var height: number = 6;
var isDone: boolean = false;
var name: string = "bob";
```

DECLARE A CONSTRAINT

Methods can be provided with information about the types which they are to process. For example, a function can be made to enforce an object with a numeric property called `index`.

```
function enforceThis(myObj: {index:
```

```
number})
{
}
```

FAIL ON COMPILE

Invoking the aforementioned method with a noncompatible parameter yields an error during transpiling. This ensures that the mistake will be detected even if the code is not run during testing.

HARNESS THE ANY

When interfacing with legacy JavaScript code, type checking must be disabled. This can be done by using the `any` variable type: the TypeScript compiler reads it as 'here be dragons'.

```
var notSure: any = 4;
```



MOOTOOLS

JQUERY HAS ESTABLISHED ITSELF AS A ONE-STOP SHOP FOR ALL KINDS OF JS GOODNESS. THIS LIBRARY WANTS TO COMPETE

MooTools provides a smorgasbord of useful functions and these will range from DOM manipulation to event management. The developers even went so far as to throw in a basic class framework that works very similar to TypeScript:

```
var Rocket = new Class({
  initialize: function(age){
    this.age = age;
  }
});
var Bisnovat = new Class({
  Extends: Rocket,
  . . .
});
```

Picking between MooTools and jQuery is - by and large - a question of taste. MooTools aims to provide useful tools to JavaScript developers, whereas jQuery promises to change the way JavaScript itself is written.



WAKANDA

NODE.JS STARTED JAVASCRIPT'S JOURNEY TO THE SERVER SIDE. THIS FRAMEWORK TIES UP SOME LOOSE ENDS

Wakanda differs from Node.js in that it offers a one-stop-shop solution for all questions related to web development. The individual modules of an application communicate via a proprietary HTTP-based protocol - it achieves more than decent performance.

The front- and backend live in a Wakanda-provided IDE, which hides many implementation details from you. This is both good and bad: creating data stores and similar classes in a graphical editor means that you can work faster due to not needing to bother with technical details. On the other hand, this precise lack of depth can become problematic when the solutions themselves become very complex.



KENDO MOBILE UI

ON MOBILE APPS, SIMILARITY IS GOOD. KENDO'S UI STACK IS A TRUE MASTER OF MIMICRY

Kendo UI sees itself as a complete solution for the client-side part of your application. Telerik provides a set of charting classes, a data-binding engine and a large selection of jQuery UI-based widgets with additional styles to choose from.

Most widgets can be created from the markup or they can be created via a constructor function. For example, an autocomplete widget can be written into an arbitrary input tag as per following:

```
$(function() {
  $("#animal").kendoAutoComplete({
    dataSource: [ "Ant", "Antelope", "Badger",
      "Beaver", "Bird" ] });
});
```

Single-page applications tend to be based on a set of three control groups: an application object that handles

page activation, and one or more pages, which consist of layouts and individual views.

Mobile-first websites start out with the declaration of an application object, which is then invoked for toggling. Forms can then be inserted via a nested div tag declaring header, footer and content panes:

```
<div data-role="view">
  <div data-role="header">Header</div>
  <a data-role="button">Click Me!</a>
  <div data-role="footer">Footer</div>
</div>
<script>
  var app = new kendo.mobile.Application();
</script>
```

FLEXIBILITY IS KEY

Many UI stack vendors can not resist the urge to act as evangelists for their favourite back-end technology.

Telerik's commercial focus protects you from such shenanigans: Kendo UI integrates itself with AngularJS and KnockoutJS; the GUI stack itself can be combined with Twitter Bootstrap.

Please keep in mind that Kendo UI is not fully open sourced. Deploying advanced features in a commercial application requires the purchase of a licence; a 30-day trial is made available for evaluation purposes.

KENDO WIDGETS

Kendo provides a set of phone-optimised widgets with ten different themes that mimic the look of Windows Phone, Android, iOS and BlackBerry. Telerik frequently changes them as the host platform's UI stacks change.

ENYO

PALM'S NEXT-GENERATION USER-INTERFACE FRAMEWORK CAN NOW BE RUN ON PCS, SMART TVS AND MOBILE PHONES

The planning process for a next-generation GUI stack for WebOS exposed a key weakness in JavaScript. Its creation process of reusable components is actually quite difficult due to the prototype-based nature of the language itself.

Enyo has addressed this problem by introducing a design pattern known as 'kinds'. Each kind realises a class, and these can then be instantiated in order to create the actual object that you may need. However, you might never actually need to create a kind of your own and this is all because Enyo ships with a large selection of svelte UI widgets.

ENYOJS.COM
twitter.com/enyojs

AMAZIUM

GETTING STARTED WITH YAML CAN BE TIRESOME. AMAZIUM MAKES IT EASY BY PROVIDING PREDEFINED STYLESHEETS

Should you choose to deploy Amazium in your application, feel free to expect an extraordinarily simple integration process. Amazium uses a system of 12-column grids for ease of setup and most elements can be deployed via a single <div> tag as long as it bears the correct class attribute.

Like most other GUI frameworks, Amazium ships with a selection of form widgets and a set of layout managers, and these will handle the widgets' arrangement. One unique feature is that of automatic image scaling as the window size changes, thereby ensuring optimal presentation across platforms.

AMAZIUM.CO.UK
github.com/OwlyStuff/Amazium

FOUR MUST-KNOW FRAMEWORKS IN 2015

EMSCRIPTEN

github.com/kripken/emscripten

GOOD FOR: Running C++ applications in a web browser of your choice

This superingenious framework lets you compile C++ code for your browser.

EXT.JS

sencha.com/products/extjs/

GOOD FOR: Creating front to end solutions with complex data access layers

ExtJS is bursting with features, which makes accessing databases a breeze.

PEBBLE.JS

bit.ly/15IH3PI

GOOD FOR: Putting your web content onto Pebble OS-based smartwatches

Most wristwatches will soon be programmable. PebbleJS mobilises web apps without a line of C.

VAADIN

vaadin.com/home

GOOD FOR: Bringing Java code to the web without total rewrites

Vaadin's Java Server technology lets you create entire sites without touching HTML5 markup.

FIVE TO FOLLOW



Standing on the shoulders of Twitterati giants can improve your skills

JEREMY ASHKENAS

twitter.com/jashkenas

Jeremy develops solutions like CoffeeScript.

THE NEXT WEB

twitter.com/thenextweb

This account discusses must-know web design topics.

WINDOWS DEVS

twitter.com/windevs

Windevs is a source of useful tips and hints for Visual Studio.

MOZILLA HACKS

twitter.com/mozhacks

Discover interesting changes to web technologies before they get implemented.

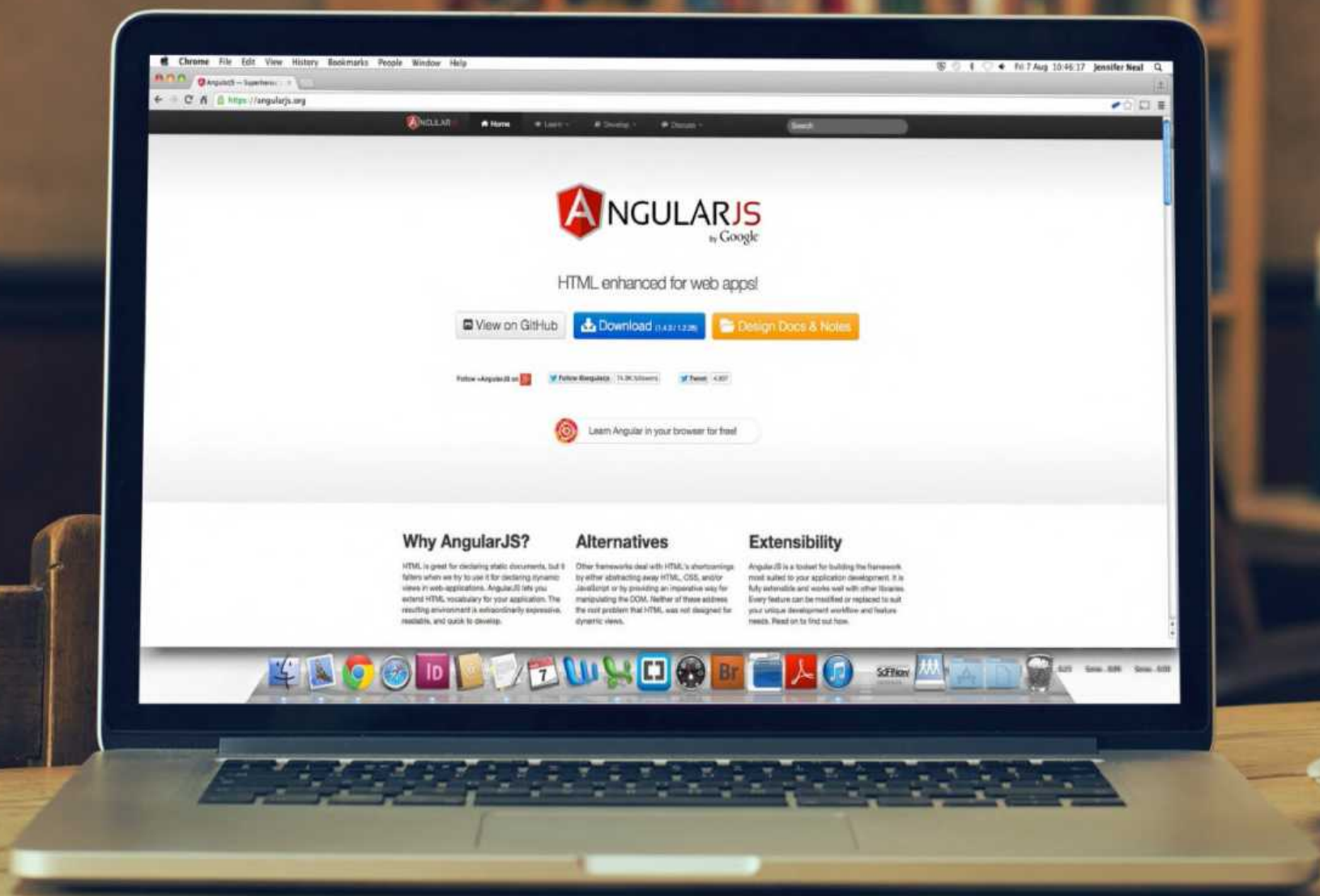
TOMI AHONEN

twitter.com/tomiahonen

An excellent source for all things mobile.



DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/bks-747





MASTER ADVANCED ANGULAR

Get your hands on an unmissable collection of tips and techniques to take your Angular code to the next level

BE AN ANGULAR EXPERT

AngularJS is a client-side MVW framework written in JavaScript, aimed at writing single-page applications (SPAs). Initially released in 2009, Angular has surfaced as one of the most popular choices for developers. It's maintained by Google as an open source project and boasts a lot of functionality built with modern web standards in mind. MVW stands for Model-View-Whatever, which gives us flexibility over design patterns when developing applications. We might choose an MVC (Model-View-Controller) or MVVM (Model-View-ViewModel) approach with it.

Angular focuses on our data, enabling our HTML to become the dynamic language that it was certainly not built for. It brings powerful concepts and ideas from various programming languages and has server-side influences that make it a top choice for web application development at present. Angular parses our declared HTML, binds it to our models that consist of plain-old JavaScript objects and keeps everything in sync with the DOM to spring our application to life. This synchronisation of DOM and JavaScript is called two-way data-binding – it is one of Angular's most powerful features amongst others such as automated DOM creation or destruction, powerful conditional logic, automated event listeners, template encapsulation, class-like components, dependency injection (DI) and promises.

Angular focuses on a fully data-driven approach to developing applications without needing to refresh models, update the Document Object Model (DOM) and other time-consuming tasks such as ironing out browser quirks and structuring our code consistently. model data can be set manually or fetched via XMLHttpRequest (XHR), which is usually a JSON resource, making it extremely easy to integrate within Controllers.

In this feature, we've packed tons of tips that originate from expert industry experience with AngularJS in the last few years, from simple time-saving to high-end performance enhancing. We're also going to create a custom Angular filter for enabling us to filter arrays of content at speed and reflect them in the UI. directives are a huge part of Angular. We'll also be looking at creating our own simple directive, focusing on structure and good practices, built-in features and more, to take away and apply to your development techniques.



REASONS FOR DIRECTIVES AND ENCAPSULATION

Modern web standards are rapidly approaching, more specifically the Web Components standard. Web

Components are an exciting new dimension of how we'll build web documents in the future, with the intention to bring a component-based approach to developing web applications. These components will comprise of many small, reusable and possibly interlinking parts.

Encapsulate in a Web Components fashion from day one and your applications will have the ability to scale with ease, be well tested, and offer flexibility and reusability. Angular lets us pretty much do all this already (and more) through its directive API, opening up this new paradigm to developers right now.

Understanding Web Components and knowing how to implement Web Components concepts in Angular today will help you succeed in writing better components overall.

So let's see how each of these specifications is defined in the Web Components standard and how Angular lets you benefit from similar features today.

Custom Elements:

What Web Components standard says:

Lets you define your own HTML elements.

What Angular lets you do already:

Angular offers the concept of directives to let you define your own custom HTML elements. You can create new HTML elements or augment existing ones. You can also add behaviour to HTML attributes.

Shadow DOM:

What Web Components standard says:

Gives you the ability to scope markup and styles in a separate DOM tree.

What Angular lets you do already:

Shadow DOM requires browser support so it's technically not possible for Angular to implement it if your browser does not support it. However Angular already provides similar benefits from a developer standpoint. Angular

enables you to encapsulate markup using a HTML template, logic using a directive controller and content using an isolated scope. It also offers transclusion, which can be thought of as a single insertion point from the Shadow DOM specification.

Used for encapsulating and scoping DOM, Angular mimics this behaviour by compiling HTML and injecting it into our DOM.

The idea behind Shadow DOM is to separate content from presentation, creating a clear distinction between where our templates contain placeholders require content, and the content itself to be parsed and injected. Shadow DOM is technically a 'DOM within DOM', it's entirely scoped and you can create as many instances of each component as you need.

HTML Imports:

What Web Components standard says:

Provides a way to include and reuse HTML documents in other HTML documents.

What Angular lets you do already:

Our components need dependencies, so we need to import them using HTML Imports. Angular actually does all this for us, and we tie our imports much closer to the component definition itself with Angular, as the HTML Imports standard actually requires us to import in the <head> of our document.

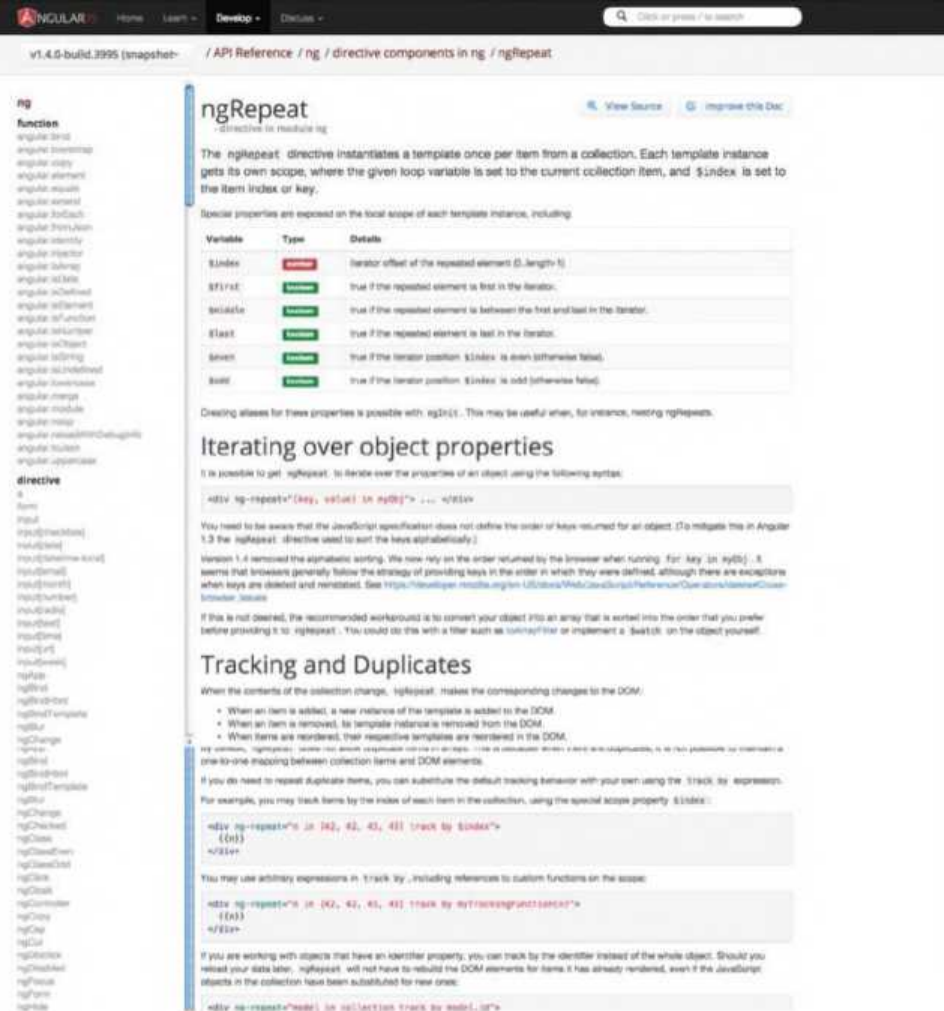
HTML Templates:

What Web Components standard says:

Enables you to define blocks of markup with the ability to inject dynamic content into.

What Angular lets you do already:

Angular enables us to use string templates or reference HTML files, enabling us to use a method best for our team, us or the project. Our templates contain no content, Angular uses {{ handlebar }} syntax for adding placeholders for where our content will be injected, compiled and finally inserted in the DOM.



REFERENCE FILTERED OBJECTS/ ARRAYS OUTSIDE NG-REPEAT

Using an ng-repeat is one of the most powerful features in Angular, enabling us to clone template portions (such as `` elements) and populate them dynamically from model data. These repeats can be used in conjunction with a filter, for instance if the user types to search for a user in a set of users.

Angular will re-render the DOM as the user types to gradually show less results until that user is found (like a fuzzy search for instance). This filtered list becomes scoped to the ng-repeat, which is fantastic, but we might want to use this filtered result elsewhere on the page, for instance we may want to show how many users were found in the results.

Here's what our initial code might look like:

```
<div ng-app="app">
<div ng-controller="MainCtrl as vm">
<input type="text" ng-model="search"
placeholder="Search emails">
<ul>
<li ng-repeat="email in vm.emails |
filter:search">
{{ email.label }}
</li>
</ul>
Emails found: {{ vm.emails.length || 0 }}
</div>
</div>
```

Let's create fake data to populate the "vm.emails" array:

```
angular
.module('app', [])
```

```
.controller('MainCtrl', function MainCtrl()
{
this.emails = [{
label: 'Item despatched'
},{
label: 'Reset your password'
}];
});
```

In the DOM, our Controller is instantiated as vm, which means that the this.emails inside of our Controller becomes vm.emails. We also have a section 'Emails found' which will contain vm.emails.length. If a user begins typing, the vm.emails.length never changes when we search, even though the DOM is updated to reflect any emails found. To get access to this, we will need to create a new \$scope variable on the ng-repeat declaration, we call it 'filterEmail' and then assign the filter value to it:

```
<li ng-repeat="email in filterEmails = (vm.
emails | filter:search)">
...
</li>
```

As this scope is created locally by the DOM, we have access to it inside our scope, which means we can change our HTML that's trying to output the search result length to this:

```
Emails found: {{ filterEmails.length || 0 }}
```

This now works perfectly, and the filterEmails property gets updated with our filtered results as the user types. This is just one example of how to use the data, but we have access to the entire filtered array if we need it.

Development

ANGULAR RESOURCES

NG-BOOK

NG-BOOK.COM

Mastering Angular is no easy task, but this book goes a long way in helping the reader become an expert. Touted as "The Complete book on AngularJS", and with over 500 pages, this book covers the framework from nearly every angle. Get your hands on this to take your skills to the next level.

ANGULARJS STYLEGUIDE

GITHUB.COM/TODDMOTTO/ANGULARJS-STYLEGUIDE

Need some assistance with AngularJS and style?

Then check out this Github repository. It provides a standardised approach for developing AngularJS applications in teams. The styleguide touches on concepts, syntax, conventions and is put together by an expert Google developer.

EGGHEAD.IO

EGGHEAD.IO

Videos add another dimension to learning, it presents what is happening in real-time with any narrative being explained. This is how Egghead.io approaches learning. It provides bite-sized training videos like 'angular-formly: Extending Types', 'Using ng-messages with ng-animate' and 'Introduction to ng-messages', all typically under five minutes long.

TO FOLLOW

John Papa

@john_papa

Angular evangelist, Pluralsight author and conference speaker

Jurgen Van de Moere

@jvandemo

Front-end architect, creator of Angular Express

John Lindquist

@johnlindquist

Egghead.io creator

Pete Bacon Darwin

@petebd

Angular 1.x lead developer and the organiser of AngularConnect

Brad Green

@bradlygreen

Google engineering director, manages AngularJS

REUSEABLE FILTERS

Filters are really powerful ways to manipulate data. A filter could parse data into a different output, such as converting millisecond dates into a human readable format, or filter content (like lists) by returning items that match certain criteria. Let's look at the Angular `.filter()` API, create a dummy email list and make an ng-repeat filter.

1. Change the drawing radius

Let's hook into the Angular module that we've made called 'app' and extend it further. Inside the filter's callback, return another function - this function is called with filter arguments and will return a filtered response.

```
angular
.module('app', [])
.filter('important', function important() {
  return function () {
    //
  };
});
```

2. Hook up the DOM

Create an expected ng-repeat with our filter intentions in place. There's an email in `vm.emails` and a pipe | which is where we declare the filter name 'important'.

```
<ul>
<li ng-repeat="email in vm.emails |
important" ng-class="{important: email.
important}">
{{ email.label }}
</li>
</ul>
```

Ng-class here adds 'important' class for emails with an 'important: true' property to help style them differently.

3. Pass in the array to be filtered

Next allow for some arguments, and name the first one 'items'. Let's 'return' the items to let the filter work, as we don't return anything, Angular won't get an array back to re-render the DOM with our updated filter.

```
angular
.module('app', [])
.filter('important', function important() {
  return function (items) {
    return items;
  };
});
```

4. Define important properties

Assuming we have data coming from our backend, we can make it available in our Controller, and pass it to our filter. Before it hits the filter, we need to make sure we have an 'important' property containing a Boolean value.

```
angular
.module('app', [])
.controller('MainCtrl', function MainCtrl() {
  this.emails = [{
    label: 'Item dispatched',
    important: true
  }, {
    label: 'Reset your password',
    important: false
  }, {
    label: 'Renew your home insurance',
    important: true
  }, {
    label: 'Welcome to Amazon!',
    important: false
  }, {
    label: 'Angels & Airwaves ticket
confirmation #402881',
    important: true
  }
];
```

```
}, {
  label: 'TravisCI test passing',
  important: false
}];
});
```

5. Return the important objects

Our filter function now only looks at returning the important emails. Create a new empty array called 'filtered', loop through the 'items' (which Angular will pass in the bound ng-repeat for us) and check if the important property is 'true'. If 'true', we use `array.prototype.push` to add that object into the filtered array, and finally return it once iteration is complete, giving us all important objects.

```
.filter('important', function important() {
  return function (items, important) {
    var filtered = [];
    for (var i = 0; i < items.length; i++) {
      if (items[i].important) {
        filtered.push(items[i]);
      }
    }
    return filtered;
  };
});
```

6. Toggle capability

We could make the filter better by adding in an optional parameter to enable us to toggle between 'All' emails and 'Important' emails in the same list. We can pass in a second argument to the DOM filter using a colon to separate arguments. Let's add a checkbox to enable us to reflect the `showImportant` model Boolean value. Once checked, `showImportant` becomes 'true', and our filter will run again with the second argument available as 'true', enabling us to decipher between whether the UI is requesting 'all' or 'important' emails.

DEPENDENCY INJECTION ANNOTATION SYNTAX

Dependency Injection (DI) in Angular is fairly simple, we just pass the dependency names that we want to the function as arguments, and Angular injects them in for us to use. Behind the scenes Angular converts our functions to a string and then it reads any dependencies that we're requesting. Note that at this point the arguments cannot be renamed.

When we minify our code (and we should), the argument names will be minified and Angular won't be able to interpret them correctly, so we need to explicitly tell Angular what dependencies we're requesting, and in what order we need them.

There are several ways to inject our dependencies to keep them safe when minified, for example by making use of the inline `$inject` syntax or array syntax. With the array syntax, we will need to pass the dependencies into our callback as an array, which will then hold the function as a final argument.

```
angular
.module('app', [])
.controller('MainCtrl', ['$scope',
'$rootScope', function MainCtrl($scope,
$rootScope) {
  //
}]);
```

With the `$inject` syntax, we can create a property on the function itself, which will offer us more flexibility if we want to make the code slightly less callback-looking and enhance readability.

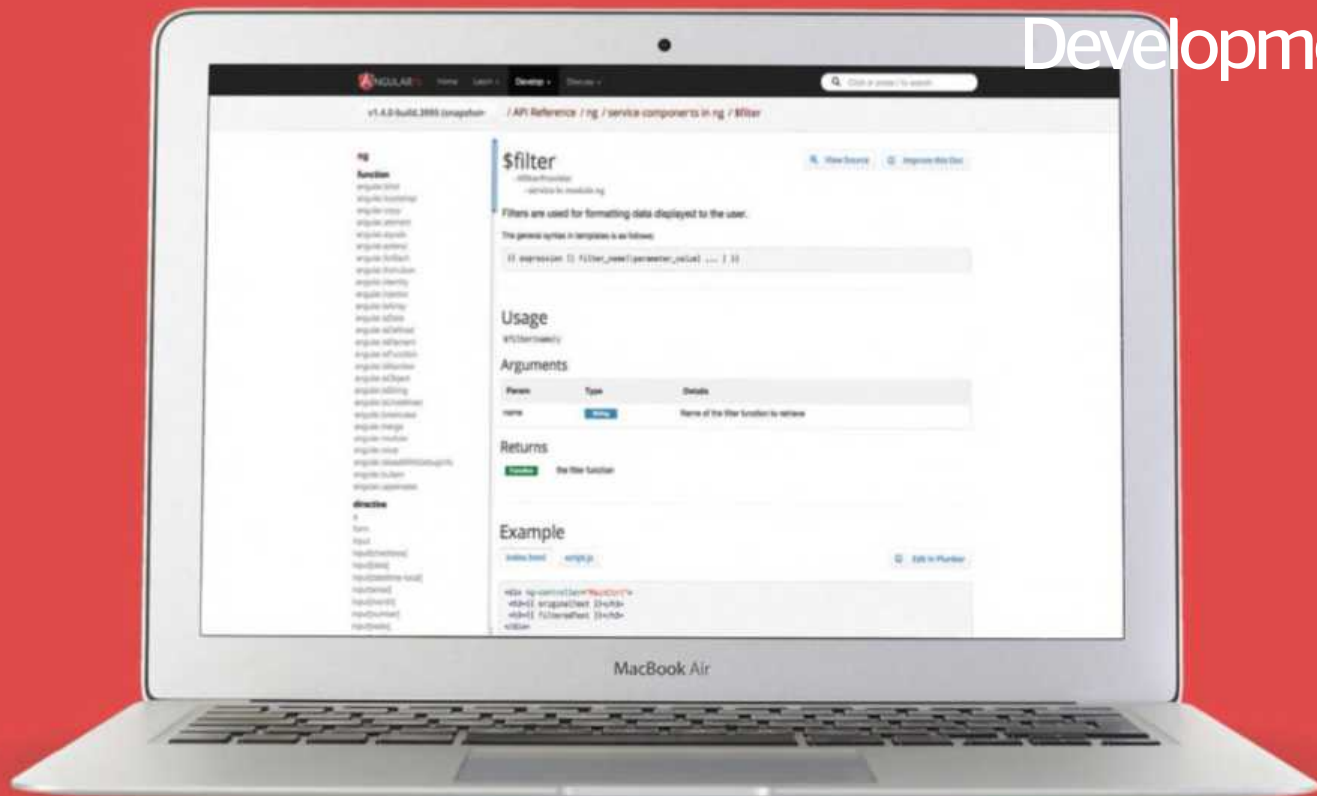
```
function MainCtrl($scope, $rootScope) {
  //
}
```

```
MainCtrl.$inject = ['$scope', '$rootScope'];
angular
.module('app', [])
.controller('MainCtrl', MainCtrl);
```

We can alternatively use a mix of both if preferred (using the array syntax but also passing a function in).

```
function MainCtrl($scope, $rootScope) {
  //
}
angular
.module('app', [])
.controller('MainCtrl', ['$scope',
'$rootScope', MainCtrl]);
```

Use `ng-annotate` to automate both of these syntaxes so we can save a lot of time typing, it's clever enough to know whether to use `$inject` or the array syntax. You can install `ng-annotate` via NPM, point it to your project files, let it compile and automatically add the dependency injection annotations for you.



```
<input type="checkbox" ng-
model="showImportant">
<ul>
<li ng-repeat="email in vm.emails |
important:showImportant" ng-
class="{important: email.important}">
{{ email.label }}
</li>
</ul>
```

7. Augment arguments

As we've declared another argument in the DOM, we can get another argument available in the JavaScript filter code, aliased as 'important'. Add an if statement to check if 'important' is passed in, otherwise return all array items.

```
.filter('important', function important() {
return function (items, important) {
if (important) {
// return only important emails
}
// if important is false, we'll return all
items
return items;
};
})
```

8. Refactor our filtering

Now we have an 'important' flag being passed in, we can move our earlier code into the 'if' statement, and run our filter when we want all our important Objects back,

otherwise we'll just assume the user requests 'all' and send the entire collection back.

```
.filter('important', function important() {
return function (items, important) {
if (important) {
var filtered = [];
for (var i = 0; i < items.length; i++) {
if (items[i].important) {
filtered.push(items[i]);
}
}
return filtered;
}
return items;
}; })
```

ONE-TIME BIND WHERE IT IS NEEDED

Angular 1.3 introduced a brilliant feature called **one-time binding**. Using one-time binding is good as it removes \$\$watchers from the \$digest cycle after they've been parsed and populated. If the data is a one-time static render then it can be bound once and removed from further \$digest loops, this will help to keep future loops much lighter and therefore faster for the JavaScript to look up any change. This feature will also enable us to remove the value from Angular \$digest cycle (which contains all our data to check states when any model values change) as soon as it becomes anything other than 'undefined'.

The \$digest cycle can become heavy and slow down our app, which isn't very helpful if we only need to render our data once, for example with a dynamically populated navigation, static list content or view titles. To specify that an expression only needs to be bound once, we can prefix the expression with '!', like '{! :name }'.



MATT GIFFORD
monkehworks.com
Consultant developer,
Monkeh Works Ltd.
@coldfumonkeh

"AngularJS is not just for single-page web applications or browser-based sites. The implementation of AngularJS within development stack of frameworks, for example, as Ionic enhances the effectiveness of PhoneGap or Cordova mobile development."

LINK FUNCTIONS FOR DOM

It may be extremely tempting to litter your Controllers with DOM manipulation, especially when integrating things such as plugins or third-party scripts and you need to set or get values. Directives are a perfect way to encapsulate any necessary presentational logic, as they include a Controller, and offer a gateway to the DOM through the 'link' function. This 'link' function gives us the root element for the directive, so that we can actually access any element inside it and bind things such as onmousedown listeners for example, which currently aren't part of the Angular core.

When using a Controller alongside a directive, Angular passes it to the 'link' function as a fourth argument, so we can run any presentational logic from callbacks to raw DOM manipulation.

Remember that inside raw DOM listeners we will need to run `$scope.$apply` to tell Angular to look at any new values. As an example (excluding most directive

properties) it is possible to pass in a Controller, aliased `$ctrl` to access presentational methods inside our raw DOM event listeners.

```
return {
  ..
  link: function postLink($scope, $element,
    $attrs, $ctrl) {
    // some code to fetch files from a made up
    drag/drop uploader
    var drop = $element.find('.drop-zone')[0];
    function onDrop(e) {
      if (e.dataTransfer && e.dataTransfer.files)
      {
        // assumes "uploadFiles" method handles the
        upload!
        $ctrl.uploadFiles(e.dataTransfer.files);
        // force a $digest cycle
        $scope.$apply();
      }
    }
    // events
    drop.addEventListener('drop', onDrop,
    false);
  } .. }
```

UNBIND \$ROOTSCOPE.\$ON LISTENERS

You'll likely find the need at some stage during your Angular career to use the internal events system, `$emit`, `$broadcast` and `$on`. These event methods are available in both the `$rootScope` and `$scope`.

If we change views in our application, Angular will destroy our Controller and thus it will destroy the `$scope` object associated with it. This is great, however, all `$scope` objects are child objects that will inherit from the `$rootScope`, and this will then persist throughout the entire application.

We will need to manually unbind `$rootScope` listeners by calling the returned closure function when the `$scope` is destroyed by listening to the `$destroy` event, otherwise when we revisit a view where the same Controller is used, the `$rootScope.$on` will be bound again. This can then cause duplicated events and also some issues with the data syncing.

```
var unbind = $rootScope.$on('fooEvent',
  function () {});
$scope.$on('$destroy', unbind);
```

With multiple events, we tend to use an array and loop through to automatically unbind.

```
[
  $rootScope.$on('fooEvent' function () {}),
  $rootScope.$on('barEvent', function () {})
].forEach(function (unbind) {
  $scope.$on('$destroy', unbind);
});
```

ABSTRACT BUSINESS LOGIC INTO SERVICES

If you've ever been tempted to use things like `$http` (Angular's XHR wrapper) inside a Controller, then it's time to understand the patterns you're using. A

Controller is a glorified ViewModel, a `$scope` object, which consists of maintaining and presenting (you guessed it) the presentational layer. Controllers should only withhold presentational logic.

Angular provides us with service, factory and provider APIs, to delegate business logic into. This makes sure our presentational layer is separated clearly from our business logic. Business logic would usually hit an API endpoint, fetch data and a Controller will make a copy of that data, pass it to the `$scope` for Angular to express what our model currently looks like.

AVOID DOM FILTERING

Using a filter in the HTML of Angular is great, but can impact performance. Filters can be easily added using a pipe inside an expression like so:

```
{{ someDateValue | filter:date }}
```

That's an example of using Angular's built-in date filter, we can however write our own filters. Filters are fairly common on ng-repeat declarations however, and can be huge performance killers.

Filters run on every `$digest` loop and will create a new array every time that it runs. Watch for changes to a specific model (such as a user typing) or a user interaction (such as a user clicking a button) and then run a filter inside the Controller's JavaScript using the `$filter` method, which will manually run your own filter upon detection of any change.



TIM STONE

fetimo.com

Lead front-end developer,
Redweb.

@timofetimo

"Updating your UI to react to changes can be a messy, hard to maintain mess. Angular abstracts this by dealing with the DOM for you. You then start to think of your applications state, which is a fundamental shift."

```
return {
  postLink: function postLink($scope, $element, $attrs, $ctrl) {
    // Some code to fetch files from a made up drag/drop uploader
    drop = $element.find('.drop-zone')[0];
    function onDrop(e) {
      if (e.dataTransfer && e.dataTransfer.files) {
        // Assumes "uploadFiles" method handles the upload!
        $ctrl.uploadFiles(e.dataTransfer.files);
        // Force a $digest cycle
        $scope.$apply();
      }
    }
    $element.on('drop', onDrop, false);
  }
};
```

\$ROOTSCOPE.\$APPLYASYNCFOR BATCHING XHR

The Angular 1.3 release saw another quietly brilliant feature, `$rootScope.$applyAsync` for intelligent XHR batching. Angular uses 'dirty-checking' to make sure all values are up to date, it does this by running what it calls a `$digest` cycle. The `$digest` can have a lot of data in, making UI response quite slow at times due to high watcher count. If we have multiple XHR calls happening around the same time, Angular is going to run a `$digest` after each of these to make sure all of its values are up to date. That's a lot of potential `$digest` loops happening around the same time. Using `$applyAsync` mitigates this issue and can result in significant performance improvements for apps making many concurrent HTTP requests. It's easy to use, so why not just drop it in to your config to see how it works?

```
angular
  .module('app')
  .config(function ($httpProvider) {
    $httpProvider.useApplyAsync(true);
  });
```

UNIT TEST ANGULAR MOCKS

Karma can run your unit tests whether in a physical browser (such as Chrome) or headless (PhantomJS). Angular mocks (from the Angular core team) extends Angular with extra functionality that you may need inside your unit tests. For example, Angular mocks makes it easy to test Controllers, directives and services (and more) by letting you mock HTTP requests, test async methods and simulate responses.

WHEN TO USE NG-BIND OVER HANDLEBAR EXPRESSIONS

Angular provides us two ways to insert text into our HTML, using the `ng-bind` or expression `{{ }}` syntax. An issue you will face if you have HTML content already on the page whilst the browser is parsing the HTML, before Angular has even loaded, is you'll see these curly expression braces, which then populate with data as Angular springs into life. To get around this we can use `ng-bind`, which will inject the value once Angular has loaded, and we'll just see an empty element which is a better experience than expression tags everywhere.

```
<!-- expressions -->
<p>{{ foo }}</p>
<!-- ng-bind -->
<p ng-bind="foo"></p>
```

Any views that are loaded with Angular routers, such as `ui-router` or `ngRouter` will compile all HTML and data before injecting the view, so you only have to cater for this issue on initial load.

“The `$digest` can have a lot of data in, making UI response quite slow at times due to high watcher count. If we have multiple XHR calls happening around the same time, Angular is going to run a `$digest` after each of these to make sure all its values are up to date. That's a lot of potential `$digest` loops happening”

FAVOUR NG-HREF ATTRIBUTES OVER HREF

Avoid 404 errors whilst the browser downloads content with expressions in the HTML. The browser will begin downloading `` rather than the parsed value (ie `app/images/niels.jpg`). Using `ng-href` will overcome this, as Angular will populate and set href attributes for us after the expressions have been evaluated.

STANDARDISE HTML BINDING SYNTAX

Angular promotes the use of their directives like `ng-click`. However `ng-click` can also be written as `x-ng-*`, `ng-*`, `ng_*` and `data-ng-*`. If you need to be standards-compliant with your attributes, use `data-ng-*`, though this is most commonly seen as `ng-*` prefixed.

NAME ANONYMOUS FUNCTIONS

If you're passing in anonymous functions into Angular's API, such as `.controller()` then name the function you are passing in. This allows `.controller('MainCtrl', function() {});` to become `.controller('MainCtrl', function MainCtrl() {});`, which really aids in stack-trace debugging in the dev tools.

BE RESPONSIBLE AND AVOID LONG API CHAINS

Aim to keep one component per file, a single responsibility. Only chain what is necessary, for instance a `MainCtrl.js` file will contain:

```
angular.module('app')
  .controller('MainCtrl', function MainCtrl() {
    ...
  });
```

DOWNLOAD AND USE BATARANG

Brian Ford (Angular core team member) launched **Batarang** for AngularJS developers. Batarang is very useful as it "extends the Chrome Developer Tools, adding tools for debugging and profiling AngularJS applications."



REUSEABLE DIRECTIVE FOR INCREMENTING OR DECREMENTING NUMBERS

This directive will use two-way data binding to take care of updating a parent model from any `$scope`, through the use of isolate scope. Encapsulating logic enables us to reuse it, and isolating `$scope` enables us to use the same directive in many different places.

1. Custom elements

Before we begin, let's create a custom element named 'number-picker', which will have an attribute called 'value' (this can be named something different). The 'expression' is just a placeholder for a model value we're passing later.

```
<number-picker value="expression"></number-picker>
```

2. Directive declaration and HTML

Next we'll create a basic directive definition, and add a template property. Directives can use templates from a string or from an external file using `templateUrl` (pointing to a relative URL). Let's set up the base directive properties and add our HTML template in.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
function numberPickerCtrl() {
}
return {
restrict: 'E',
scope: {},
template: [
'<div class="number-picker">',
'<div class="number-picker__button number-
picker__button--decrement"></div>',
'<input class="number-picker__input">',
```

```
'<div class="number-picker__button number-
picker__button--increment"></div>',
'</div>'
].join(''),
controllerAs: 'vm',
controller: numberPickerCtrl
};
});
```

3. Isolate scope binding

We need to pass the updated value in the directive back to where it was initialised after changes have taken place, this lets us keep other data models in the parent Controller updated. Pass the value in via the scope property, creating a new object to initiate 'isolate scope', and creating a property name called 'value'. We can use `=` as the value, which tells Angular to bind the value two ways, back up the parent Controller.

```
.directive('numberPicker', function
numberPicker() {
...
return {
...
scope: {
value: '='
},
...
};
});
```

4. Controller methods

We need to implement methods to increase or decrease our number inside the Controller. As we're using the `controllerAs` syntax, we need to attach these to the

function instance using the 'this' keyword rather than injecting `$scope`.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
function numberPickerCtrl($scope) {
this.increment = function () {
};
this.decrement = function () {
};
}
...
});
});
```

5. Model manipulation

As our value is passed through `$scope`, we access it using `$scope.value` to pass back up to the parent model and not with `this.value`.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
function numberPickerCtrl($scope) {
this.increment = function () {
++$scope.value;
};
this.decrement = function () {
--$scope.value;
};
}
...
});
```


6. Controller methods

Now we've got Controller methods to modify our value, we need to make sure our HTML template can talk to those methods through Angular's built-in directives, we need `ng-click`. As we're using `controllerAs`, the Controller is instantiated as a reference object in the DOM as `vm` (ViewModel) so we can access our public `increment` or `decrement` methods through the `vm` object.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
...
return {
...
template: [
'<div class="number-picker">',
'<div class="number-picker__button number-
picker__button--decrement" ng-click="vm.
decrement()"></div>',
'<input class="number-picker__input"
ng-model="value">',
'<div class="number-picker__button number-
picker__button--increment" ng-click="vm.
increment()"></div>',
'</div>'
].join(''),
...
};
});
```

7. Add \$watch to validate input

As we enable the user to type into the input, we need to ensure that they only enter a number. We can use `$watch` to watch changes to the value, and reset it back to its old value if it's anything other than a number. We can use a simple Regular Expression to test if anything isn't a number, and revert it back to the previous value. This is just basic validation, a more comprehensive solution may fit your needs in a production application. We could use `ng-pattern`, but it still enables users to type anything, and Angular will refuse to set the model unless the element matches the pattern.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
function numberPickerCtrl($scope) {
$scope.$watch(function () {
return $scope.value;
}, function (newVal, oldVal) {
if (! /^[0-9]+$/.test(newVal)) {
$scope.value = oldVal;
}
});
...
}
...
});
```

8. Link function keyboard input

We're also going to let the user press up or down to increment or decrement the number. We can do this via the `'link'` property on the directive, which gives us access to the root node of the directives instance, as well as the `$scope` and any attributes it has. Set up placeholders for handling up and down arrow keys through the `keyCode` property on the events passed to us.

```
angular
.module('app', [])
.directive('numberPicker', function
numberPicker() {
function numberPickerCtrl($scope) {
$scope.$watch(function () {
return $scope.value;
}, function (newVal, oldVal) {
if (! /^[0-9]+$/.test(newVal)) {
$scope.value = oldVal;
}
});
this.increment = function () {
++$scope.value;
};
this.decrement = function () {
--$scope.value;
};
function numberPickerLink($scope, $element,
 attrs) {
function handleKeyUp(e) {
if (e.keyCode === 38) {
}
if (e.keyCode === 40) {
}
}
$element.find('input').on('keyup',
handleKeyUp);
}
});
```

9. Pass in Controller

You can also pass through the Controller into the `'link'` function as the fourth function argument (aliased as `$ctrl`) to make a clear distinction between where we hold presentational logic and DOM logic. We should ideally only use `'link'` for raw DOM communication. This means we call our existing methods to manipulate the value, without polluting our Controller with DOM logic – a big antipattern in Angular. We need to run `$scope.$apply()` after raw DOM APIs to inform Angular of changes and to update any values that we updated ourselves.

10. Final touches

Let's see what everything looks like together and check `FileSilo` for the full code. Using the expression `"{{ vm.value }}"` we can get updates as the user types which are instantly reflected back up to the parent Controller, as it's the property delegated to the directive, manipulated and passed back up.



TODD MOTTO
@toddmotto
The Google developer expert and director of front-end engineering at Mozio gives us five Angular tips.

USE DIRECTIVE CTRL STRING METHODS

Instead of assigning a Controller and using an anonymous/assigned function, use a string-based approach. This references a Controller created using the `.controller()` method and makes the Controller reusable elsewhere in the application and testing much easier.

APPLY NG-CLOAK CLASS NAME

Elements with conditional statements such as `ng-if`, `ng-show` or `ng-hide` are sometimes visible until Angular has evaluated the expression. We can use `ng-cloak` attribute, which is essentially styled as `display: none;`. After Angular has evaluated expressions, it removes all `ng-cloak` attributes leaving the element in its correct conditional state.

ROUTING 'RESOLVE' PROPERTY

Ensure asynchronous operations are complete by the time Angular switches views, this ensures all data is fetched prior to rendering the view. The `resolve` property sits on both the Angular router (`ngRouter`) and `ui-router`. We can pass in services that resolve a promise, these resolved promises are then given to us in the Controller through dependency injection.

\$SCOPE IS JUST A VIEWMODEL

Treat your `$scope` as a ViewModel at all times, never use DOM logic or even think to include jQuery! All a Controller should do is make a copy of data that's provided to it from a service, and express that model through Angular's `ng-*` directives. Angular Controllers are our presentational layer, and shouldn't be mixed with DOM logic or business logic.

TESTABLE DIRECTIVES THROUGH ABSTRACTION

Lessen the impact of testing directives by abstracting as much as you can into Controllers and services (providing the architectural pattern is correct for your code). This enables us to bullet-proof our workflow by writing tests for the Controller and service independently, and it also enables our directive test to be smaller and less data focused.



DOWNLOAD TUTORIAL FILES
www.filesilo.co.uk/bks-747

Construct an image gallery with CSS and AngularJS

Combine AngularJS and CSS transitions to create an impressive image gallery user interface quickly **tools | tech | trends** AngularJS, JavaScript, HTML, JSON



Image gallery UIs are often required within clients' website projects, and having a reusable, modular solution can save valuable time in your builds. There are many jQuery gallery plugins available, but these are often quite large. These plugins rely on jQuery's DOM

first approach to UI interactions, requiring your content

data to be held and referenced from the DOM. AngularJS separates your data and presentation so the UI can load and run superfast and any functionality is bound directly from your DOM.

This tutorial will cover the basic application setup, creation of a directive with its own HTML template, and the setting of data for that directive within a controller. Angular directives do more than just

manipulate the DOM, they can also receive arguments through HTML attributes, enabling config option adjustment for your resultant UI. In addition the CSS3 transition property will be used for efficient transitions. After completing this tutorial you will be able to take this structure and approach to build up a range of reusable directives for your common interactive components.

1. Create the main HTML page

Start by adding a reference to AngularJS from the Google CDN to your HTML page. Then add a reference to your application on the body tag using an attribute as shown. If you use the data-ng- prefix for your Angular attributes opposed to 'ng-' your HTML will be valid.

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.
w3.org/1999/xhtml">
<head>
<meta charset="utf-8" />
<title>Angular Gallery App</title>
<link href="css/styles.css" rel="stylesheet"
media="screen" />
<script src="http://ajax.googleapis.com/
ajax/libs/
angular.min.js"></script>
</head>
<body data-ng-app="galleryApp">
</body>
</html>
```

2. Create the core application

In your scripts folder create a file named 'myGallery.js', this file will define your core application and any dependencies will be added to your application here. Ensure the namespace defined here matches the namespace set within the 'data-ng-app' attribute in the HTML. Reference this file within your HTML.

```
<head>
<meta charset="utf-8" />
```

```
<title>Angular Gallery App</title>
<script src="http://ajax.googleapis.com/
ajax/libs/angularjs/1.3.10/angular.min.
js"></script>
<script src="scripts/myGallery.js"></script>
</head>
//scripts/myGallery.js
var myGallery = angular.module('galleryApp',
[]);
```

3. Add your CSS

Add references to your CSS files in your main HTML page. Within the tutorial files there are both 'normalize.css' and 'styles.css', feel free to work with these or your own presets.

```
<link href="css/normalize.css"
rel="stylesheet" media="screen" />
<link href='http://fonts.googleapis.com/
css?family=Poiret+One' rel='stylesheet'
type='text/css' />
<link href="css/styles.css" rel="stylesheet"
media="screen" />
```

4. Add a directive

Now add a directives folder in your scripts folder and create a file named 'galleryDirective.js'. In this file create a new angular.module and add a 'myGallery' directive as shown. A directive returns a range of properties, for now set restrict to 'A' (it makes the directive available as an attribute) and templateUrl as shown.

```
angular.module('gallery.directive', [])
```

```
.directive('myGallery', function () {
"use strict"
return {
restrict: 'A',
templateUrl: "scripts/directives/templates/
gallery.html"
}
});
```

5. Create your directive template

Within the directives folder add a folder named 'templates' with a file named 'gallery.html' and put in a piece of sample HTML in this file. Now add a reference to your new directive within your main HTML page and add the directive as a dependency in your core app file as shown. Your directive template should be rendered within your HTML page in the browser.

```
<body data-ng-app="galleryApp">
<div data-my-gallery></div>
</body>
//scripts/myGallery.js
var myGallery = angular.module('galleryApp',
['gallery.directive']);
//scripts/directives/templates/gallery.html
<h2>My New Gallery</h2>
```

6. Create a controller

To add the data for the gallery we will use a controller to hold it within a \$scope object. Within your scripts folder add a folder named 'modules' with a file named 'galleryModule.js'. Within this file create a controller and set your title text within the \$scope object.

MY GALLERY

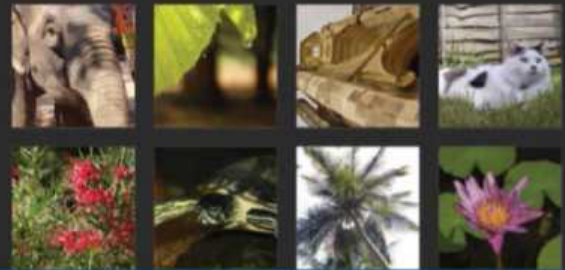
<Above>

• You should see your Gallery Title displayed through your directive. Any text value can be rendered using simple handlebars syntax

**<Above>**

• Arrays held within \$scope can be handled just as in any JavaScript syntax. In this case we select the first item using [0]

MY GALLERY

**<Above>**

• Angular repeaters are very useful for many UI scenarios, the context of each object within an array is the scope within a repeater

<Left>

• Box-shadow can be a really useful CSS property. It can add great depth to your designs and, if combined with pseudo elements, can produce some interesting effects

```
angular.module('gallery.module', [])
.controller('galleryCtrl', ['$scope',
function($scope) {
"use strict";
$scope.galleryTitle = "My New Gallery";
}]);
```

7. Use your new controller

Add your controller as a dependency to your core app file, then reference your controller within your main HTML page. Extend your 'galleryDirective.js' file with the 'scope' property. This will isolate the scope of the directive and enable you to reference your 'data-my-title' attribute.

8. Add a controller to your directive

Now the scope of the directive is isolated add a controller. This controller will hold any functionality directly tied to the gallery interface. The controller accepts arguments for the current scope, the HTML element the directive is bound to, and any attribute on that element. Then reference your new 'title' property and set in the scope within your directive Template.

```
scope: {
title: "=myTitle"
},
controller: function($scope, $element,
$attrs){
}
//scripts/directives/templates/gallery.html
<h2>{{title}}</h2>
```

9. Add some images

Now the structure of the application is set up you can add some images. Add image files (both full size and thumbnail) to an images folder. Then create a JSON object within your 'galleryModule.js' file, include both the main and thumbnail paths to each image, along with a short description. This data could be generated from a web service provided by a CMS platform.

10. Pass your images to your directive

Now use the same technique as with the 'galleryTitle' property to pass the images array to your directive, using a new attribute named 'data-my-images'. Extend the scope of your directive to set these images and make them available to your gallery.

```
<body data-ng-app="galleryApp" data-ng-
controller="galleryCtrl">
<div data-my-gallery data-my-
title="galleryTitle" data-my-
images="galleryImages" class="gallery"></div>
</body>
//scripts/directives/galleryDirective.js
scope: {
title: "=myTitle",
images: "=myImages"
}
```

11. Build your thumbnails

To display your thumbnails you can use an Angular

Why use CSS animations?

If we were building this gallery with jQuery we could use the fadeIn and fadeOut methods to animate the main image appearing. CSS transitions perform much better in browsers and are easy to adjust.

repeater. This is a core Angular directive used for repeating a block of HTML based on an array of data. Add an unordered list to your directives template file, with 'data-ng-repeat' bound to the LI. You should now see your thumbnails in your gallery.

```
<h2>{{title}}</h2>
<ul class="inline">

<li data-ng-repeat="image in images"></li>
</ul>
```

12. Use an image selection method

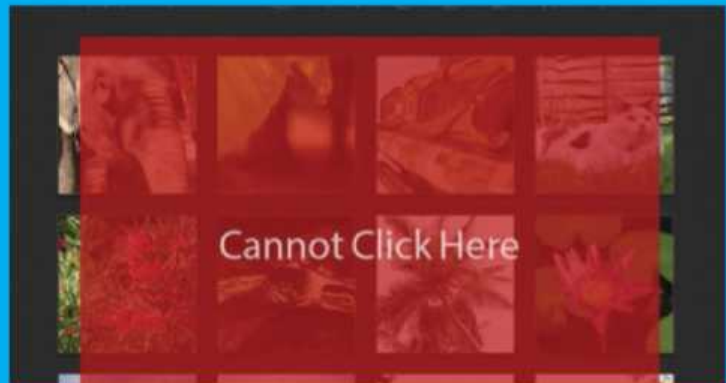
Now add a 'data-ng-click' attribute to the repeated LI item calling a method named 'selectImage' and passing the 'image' property into it.

Then add a new function to the directive controller to receive this image data. Once received you can log out the data and check in your browser console to check the correct image is being passed.



<Above>

- The transition property is great to add efficient animation transitions to your UI. It is also really quick to experiment with



<Above>

- When using opacity to hide HTML elements the browser will still behave as if they are there. This can block click events on lower elements



<Above>

- Incorporating z-index into the CSS transition gives you more control over the hidden element. Chaining transitions can be really useful



<Above>

- Try visiting css3gen.com/box-shadow. It is a great tool for creating box shadows as well as a host of other CSS3 effects

File separation of your application

The folder structure within this tutorial may seem excessive, but when integrated with a larger application, a well-defined structure makes maintenance and extension very easy to manage.

13. Add a main image

Within your directive template file `gallery.html`, add a div to hold a main image enabling you to display the larger version of the selected image. Then create a new `$scope` property within your directive controller, setting `$scope.mainImage` to the first item within the `$scope.images` array. Then change this `$scope.mainImage` within the `$scope.selectImage` function.

```
<h2>{{title}}</h2>
<div class="mainImage">
  
</div>
<ul class="inline">
```

```
<li data-ng-repeat="image in images"
  data-ng-click="selectImage(image)"></li>
</ul>
//scripts/directives/galleryDirective.js
controller: function($scope, $element,
  $attrs){
  $scope.mainImage = $scope.images[0];
  $scope.selectImage = function(image){
    $scope.mainImage = image;
  }
}
```

14. Hide the main image

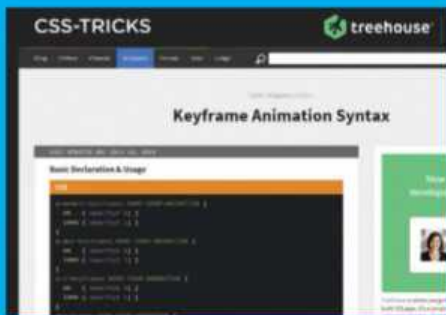
Set the `'mainImage'` div as hidden in your CSS and then create a class of `'show'` to display it. Now change the class attribute on this div to a `'data-ng-class'` attribute. You can write a shorthand if statement into this attribute and bind the class to a new `'$scope.showMainImage'` property within your directive's controller.

15. Show when selected

Within your directive controller set the `'$scope`.

`showMainImage'` property to `'false'` as default. Then set this property to `'true'` within the `selectImage` function. Create a new function named `'$scope.close'` to set this property back to `'false'`. Now add a close link into div. `mainImage` within your HTML and call the `'close'` method using a `'data-ng-click'`.

```
<div data-ng-class="showMainImage ?
  'mainImage show' : 'mainImage'">
  <a href="#" class="close" data-ng-
    click="close()">Close</a>
  
</div>
//scripts/directives/galleryDirective.js
controller: function($scope, $element,
  $attrs){
  $scope.mainImage = $scope.images[0];
  $scope.showMainImage = false;
  $scope.selectImage = function(image){
    $scope.mainImage = image;
    $scope.showMainImage = true;
  };
  $scope.close = function(){
```



<Above>

- To create more complex animation sequences try using keyframes in CSS. This is a good place to start css-tricks.com/snippets/css/keyframe-animation-syntax

```
$scope.showMainImage = false;
};
}
```

16. Handle default events

As the close functionality is now bound to an anchor link with a # href attribute, the browser window will scroll to top when clicked.

You can use the preventDefault method to stop this. Simply pass \$event to your 'close' method and then call preventDefault() on it within your directive controller.

```
<a href="#" class="close" data-ng-
click="close($event)">Close</a>
//scripts/directives/galleryDirective.js
$scope.slide = {
  $scope.close = function(e){
    e.preventDefault();
    $scope.showMainImage = false;
  };
};
```

17. Add some depth

Your gallery may be looking quite flat at this point, to add some depth to your gallery you can use CSS3 box-shadows on both the thumbnails and main image. Create a shadow class within your CSS and apply it to the thumbnail images. Also add a larger box-shadow to the main image.

```
.shadow {
  -webkit-box-shadow:0 4px 4px rgba(0, 0, 0, 0.3), 0 0 40px rgba(0, 0, 0, 0.1) inset;
  -moz-box-shadow:0 4px 4px rgba(0, 0, 0, 0.3), 0 0 40px rgba(0, 0, 0, 0.1) inset;
  box-shadow:0 4px 4px rgba(0, 0, 0, 0.3), 0 0 40px rgba(0, 0, 0, 0.1) inset;
}
.mainImage img {
  -webkit-box-shadow: 0 0 80px #000;
  -moz-box-shadow: 0 0 80px #000;
  box-shadow: 0 0 80px #000;
```

```
}
```

18. Position the main image

Within the CSS file provided with this tutorial the entire gallery is centred, however centring the absolutely positioned main image takes a little extra CSS.

You may have used this technique in the past, but it is often forgotten and if you haven't used it before it can save you lots of CSS hacking.

```
.mainImage {
  position: absolute;
  top: 130px;
  left: 50%;
  margin-left: -500px;
  width: 1000px;
}
}
```

19. Add some animation

To add an animation to your main image appearing you can use the CSS3 'transition' property. This will only work with CSS properties that have numerical values, so you will need to switch from using 'display' to 'opacity' to hide/show the main image. Then add a transition property to .mainImage to animate this change over one second.

```
.mainImage {
  position: absolute;
  z-index: 10;
  display: block;
  top: 130px;
  left: 50%;
  margin-left: -500px;
  opacity: 0;
  width: 1000px;
  -webkit-transition: opacity 1s;
  -moz-transition: opacity 1s;
  transition: opacity 1s;
}
.mainImage.show {
  opacity: 1;
}
}
```

20. Use pointer-events

You may notice that you cannot click on any of the thumbnails that are overlapped by the main image. This is because even though the main image is hidden with opacity, the browser interprets it as being an active HTML element, therefore blocking any click events. If you add 'pointer-events: none;' to .mainImage in your CSS this should be fixed.

```
.mainImage {
  position: absolute;
  z-index: 10;
  display: block;
  top: 130px;
```

```
left: 50%;
margin-left: -500px;
opacity: 0;
width: 1000px;
-webkit-transition: opacity 1s;
-moz-transition: opacity 1s;
transition: opacity 1s;
pointer-events: none;
}
}
```

21. Fix the close button

The only problem with using pointer-events is that they are inherited by child HTML elements. This means the close button no longer works. An alternative approach is to chain transition properties, set the z-index property on .mainImage to zero and then to ten on mainImage.show. Now you can add z-index as a second transition value and your animation should work fine.

```
.mainImage {
  position: absolute;
  z-index: 10;
  display: block;
  top: 130px;
  left: 50%;
  margin-left: -500px;
  opacity: 0;
  width: 1000px;
  -webkit-transition: opacity 1s, z-index 1s;
  -moz-transition: opacity 1s, z-index 1s;
  transition: opacity 1s, z-index 1s;
}
.mainImage.show {
  opacity: 1;
  z-index: 10;
}
}
```

22. Reuse your directive

Now you will have a gallery directive that is simple to integrate with any website build. Just ensure the gallery directive file is injected into your application and then you can use the attribute 'data-my-gallery' wherever you need to. If you need to add further options to extend and configure your gallery you can add more to the scope of the directive itself.



<Above>

- There is a wealth of insight and information available if you want to learn to develop with this framework. For more information, check out www.angularjs.org

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Code validation into forms with ngMessages

Learn how to create a form with reactive user feedback using Angular and the ngMessages directive



Forms are the gateway between your website and the user's life. Often forms capture sensitive data like your address, telephone number, card details and so on.

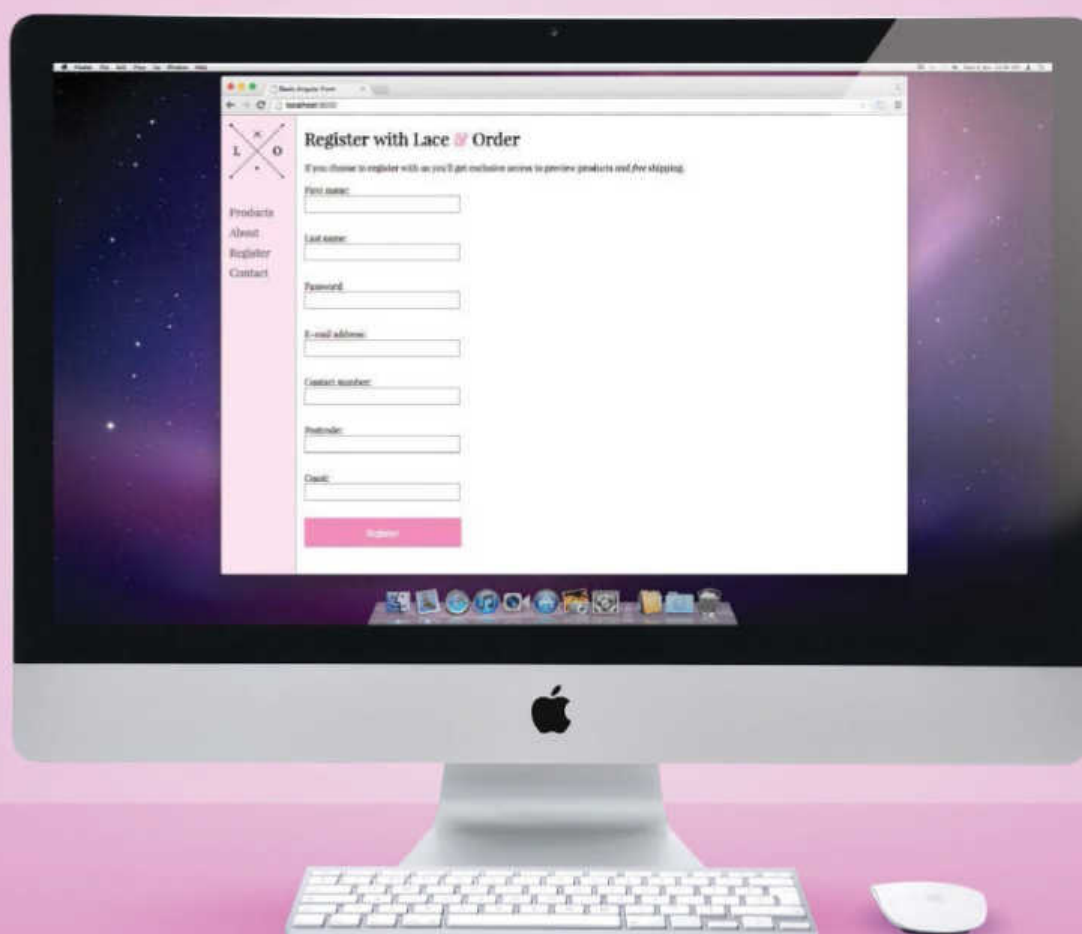
Filling in forms is annoying so as web developers we want to try and make this process as pain-free as possible. One way we can do this is to show helpful, timely messages to the user to let them

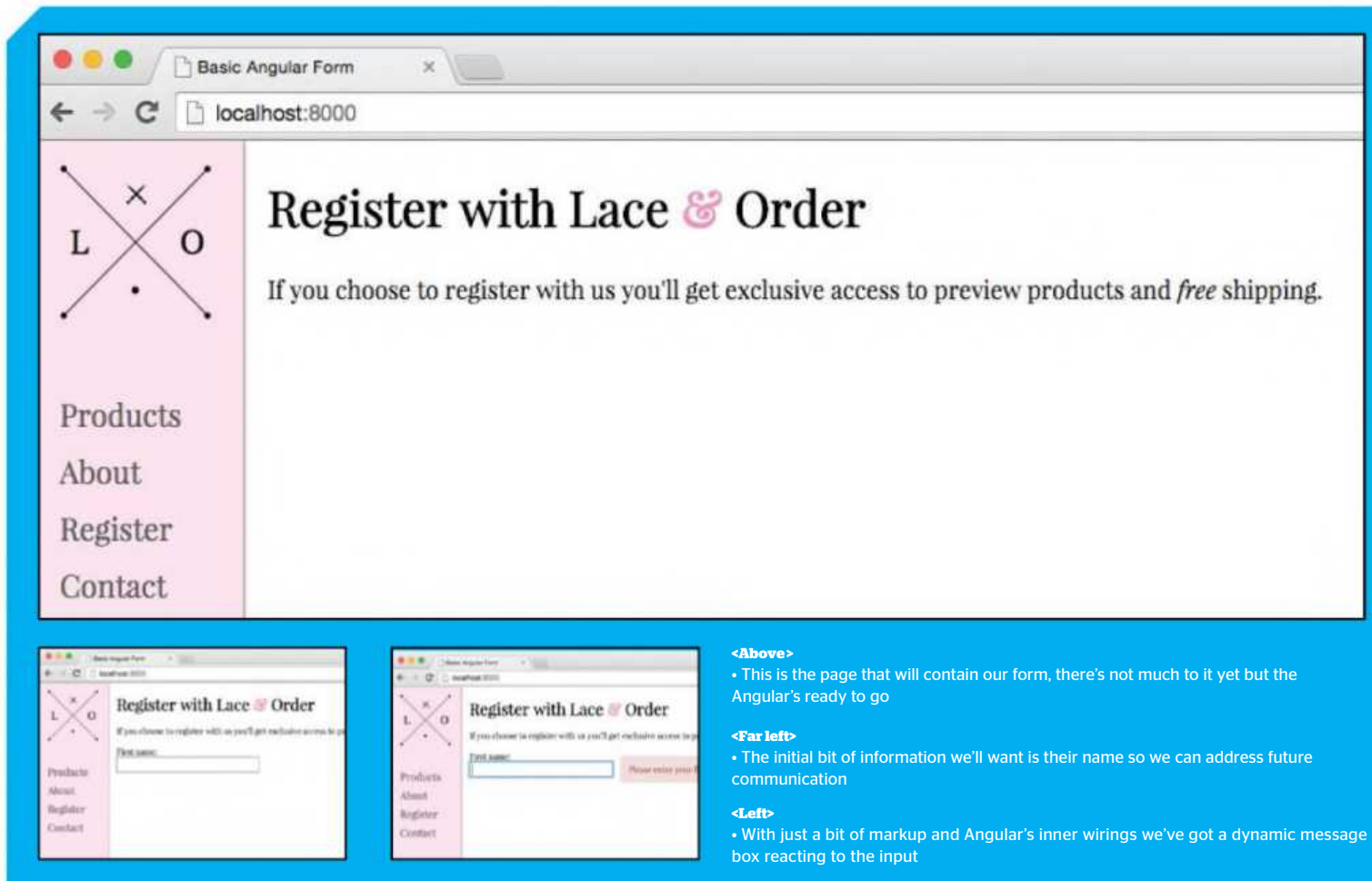
know if they've made a mistake. These are essential to reducing friction in forms and aiding understanding. We also want to try and make them as accessible as possible so we will include ngAria, too.

Angular's ngMessages module provides developers with an easy way to present messages to a user. It's described as being "designed to handle the complexity, inheritance and priority sequencing based on the order of how the messages are defined in the template". You

could use ngIf to toggle the visibility of messages but this can be complicated when there are many conditions. NgMessages was introduced in Angular 1.3 so it doesn't work with Internet Explorer 8 or below.

In this tutorial you'll learn how to create a form and use Angular to validate it. Our example will be a registration form for the fictitious 'Lace & Order' shop. It'll feature custom validation directives, default messages and utilising ngModel to its full ability.





1. Install Angular and modules

First off we're going to add `ngMessages` to our project and our preferred method of adding this in is to use Bower (go ahead and visit bower.io/#install-bower if you haven't got it installed already). As well as the Messages module we're also going to make use of the `Animate` and `ARIA` modules.

```
$ bower install angular-animate angular-aria
angular-messages
```

2. Include script files

Create a file called 'index.html' and then write the markup required for a skeleton HTML page (with the `<head>`, `<body>` and so on). Next we'll import the JavaScript files that Bower had fetched for us just before the closing body tag. Even though we're using the minified versions here, the browser will map to the full source files.

```
<script src="bower_components/angular/
angular.min.js">
</script>
<script src="bower_components/angular-
animate/angular-animate.min.js">
</script>
<script src="bower_components/angular-aria/
angular-aria.min.js">
</script>
<script src="bower_components/angular-
messages/angular-messages.min.js">
</script>
<script src="scripts/app/app.js">
</script>
<script src="scripts/app/modules/
registerModule.js">
</script>
<script src="scripts/app/controllers/
registerController.js">
```

Directive names

In real-world applications you should always prefix your directive names with a couple of letters. This differentiates them from standard HTML elements.

```
</script>
```

3. The formApp module

The module that will contain all the application-wide dependencies is called `formApp`. As well as the Angular modules that we included in the previous step `registerModule` is also included. This will include the dependencies specific to the registration form.

```
angular.module('formApp', ['register.
module', 'ngAria', 'ngMessages',
'ngAnimate']);
```

4. The register module

When we write custom directives you'll also have to add them as a dependency to this module. It's easy to forget to do this and find yourself spending

As well as the Angular modules that we included in the previous step `registerModule` is also included

<Above>

- The telephone error message is shown when the input doesn't match a REG_EXP pattern using ngPattern

<Below>

- The postcode validation error message appears when we input an incorrect postcode meaning our custom directive is working

<Right>

- The default messages are now included in the template unless they're overridden by one further up in the markup

Debugging \$errors

A useful way to get a quick understanding of an object in real-time is to output it onto the page:

```
<pre>{{form.postcode.$error | json}}</pre>
```

 This will show the object and format it as JSON.

minutes figuring out why your directive isn't working. So don't forget to save yourself time!

```
angular.module('register.module',
  ['register.controller']);
```

5. Register controller

The controller for our registration form is very bare bones. So what we'll do is we'll add some specific uses to it later (namely to add dynamic text). Behind the scenes it will do the internal wiring to ensure our form works as we expect it to.

```
(function() {
  'use strict';
  angular.module('register.controller', [])
    .controller('registerController', ['$scope',
```

```
function ($scope) {}]);
})();
```

6. Create the form tag

The form tag is going to do a lot of work for us. It's going to be the root of our application and also the controller. NgSubmit is evaluated each time the form is submitted (either by the user clicking the submit button or by pressing the Enter key).

```
<form name="register"
  data-ng-app="formApp"
  data-ng-controller="registerController"
  data-ng-submit="submitForm(register)">
```

```
<h1>Register</h1>
</form>
```

7. The ng-attr directive

A couple of other attributes need to be added to the form tag. Ng-attr-novalidate is an Angular directive which will only apply the novalidate attribute if the expression evaluates to true. We want to disable the native HTML5 validation if the Angular app has successfully loaded.

```
<form data-ng-attr-novalidate="{{true}}"
```

```
data-ng-cloak>
</form>
```

8. First name input

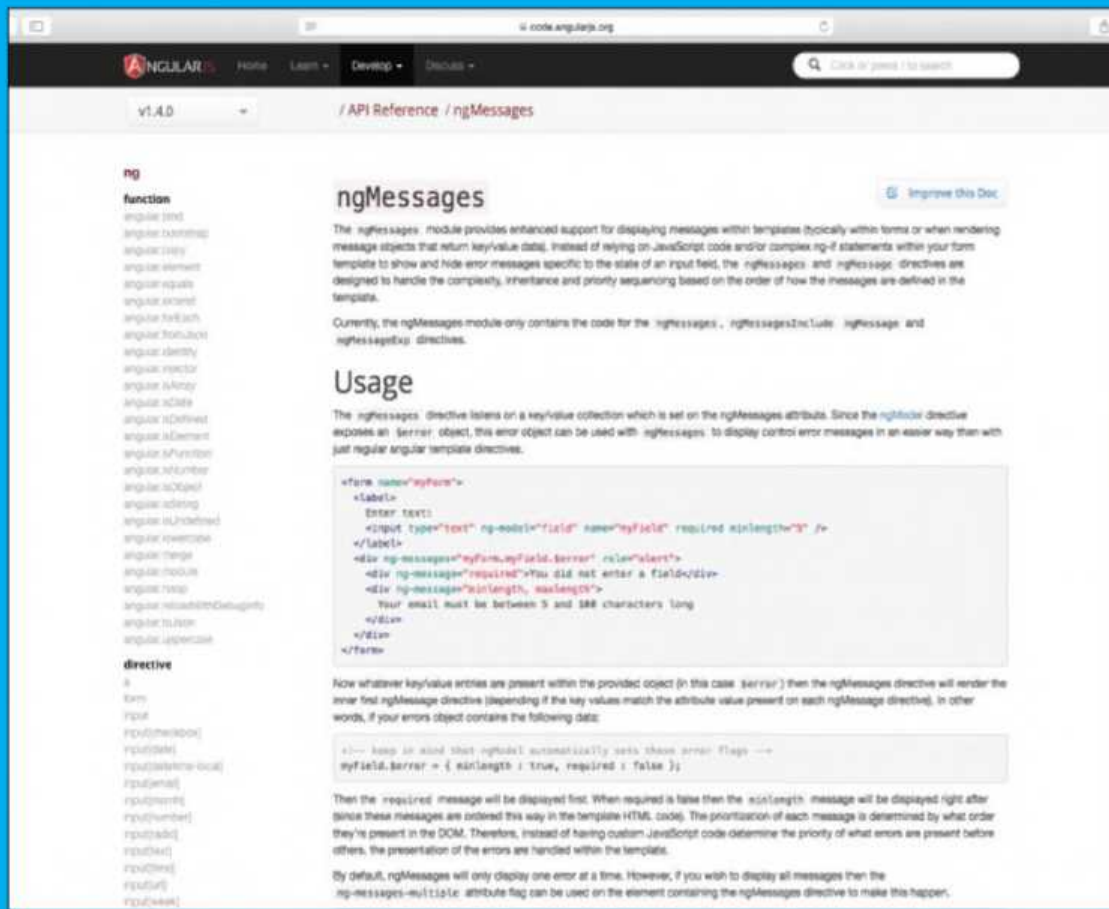
The first piece of data that we'll capture from the user is their first name. NgModel is used to bind data to the input field. This means that when the data changes, the model is immediately updated. The rest of the markup is your regular HTML5.

```
<div class="row">
  <label for="firstName">First name:</label>
  <input id="firstName" name="firstName"
    data-ng-model="fields.firstName" required
  />
</div>
```

9. ngMessages first name

Here's our first example of ngMessages in action. Ng-messages takes an Angular expression which evaluates to a key/value object. This behaviour is just like a switch statement. In the example that we have here, the object evaluated is the \$error property from ngModel in the firstName field.

```
<div class="errors" data-ng-
```



Utilise ngMessage expressions

NgMessages also supports dynamic messages, that is, if you don't know the name of the message that might be fired you can check it with a message expression. In this example we've got an array in the controller called `errorMessagees` which contains a list of objects. These objects represent different error states and their corresponding text.

```
<div data-ng-
messages="register.
test.$error" role="alert">
<div data-ng-
repeat="errorMessage in
errorMessagees">
<div data-ng-message-
exp="errorMessage.type">
<p>
```

The `ngMessages` documentation has pages on each directive: docs.angularjs.org/api/ngMessages.

```
messages="register.firstName.$error">
```

```
<!-- next step -->
</div>
```

10. ngMessages directive

NgMessages does a great job of toggling specific messages but we also need to determine when to show them.

Otherwise they'll appear before the user has touched the form! The messages should only appear if the field is dirty (if it has been filled in) or the user submits the form.

```
<div class="errors" data-ng-
messages="register.firstName.$error"
data-ng-if="register.firstName.$dirty ||
```

```
register.$submitted">
```

```
<!-- next step -->
</div>
```

11. The required property

Within the `ngMessages` directive, we can now go ahead and list specific error conditions to show. This example here has a simple paragraph tag, but you can use any element as well as place any markup that you want inside of it. 'Ng-message="required"' looks for the 'required' property on the object that we passed above. If true, then it'll show:

```
<p class="error" data-ng-
message="required">Please enter your first
name.</p>
```

12. Style error messages

Our error messages deserve a bit of love so let's apply some CSS to them. The most important parts to note are the opacity and transition properties. NgAnimate will use them in conjunction with other classes to ease them in rather than abruptly appearing.

```
.error
{
color: #a94442;
padding: 0.875em;
background-color: #f2dede;
width: 16em;
border: 0.125em solid #ebccd1;
transition: 0.5s linear opacity;
opacity: 0;
margin: 0;
}
```

13. Submit the form

To allow the user to submit the form we're going to need a submit button! This calls the submit handler that was hooked up at the top of the form. However the user submits the form - be it by click, programmatically or pressing Enter - it calls the same function.

“ This example here has a simple paragraph tag, but you can use any element as well as place any markup that you want inside of it ”


```
<div class="row">
<input type="submit" value="Register" />
</div>
```

14. Telephone input

Let's take the concepts we've looked at and apply them to a new field, a contact number. This time ngPattern allows us to match what the user types to a regular expression. This regular expression checks that only numbers are typed.

```
<div class="row">

<label for="tel">Contact number:</label>
<input id="tel" name="tel" data-ng-
model="fields.tel" type="tel" data-ng-
pattern="^[0-9]+$/" required />
<!-- next step -->
```

```
</div>
```

15. Telephone error messages

This time we're going to use the same technique as before to show a message. As well as 'required' there will also be a 'pattern' property as ngPattern has been used. You can chain properties together on the same error message to prevent repeating yourself.

```
<div class="errors" data-ng-
messages="register.tel.$error" data-ng-
if="register.tel.$dirty ||
register.$submitted" role="alert">

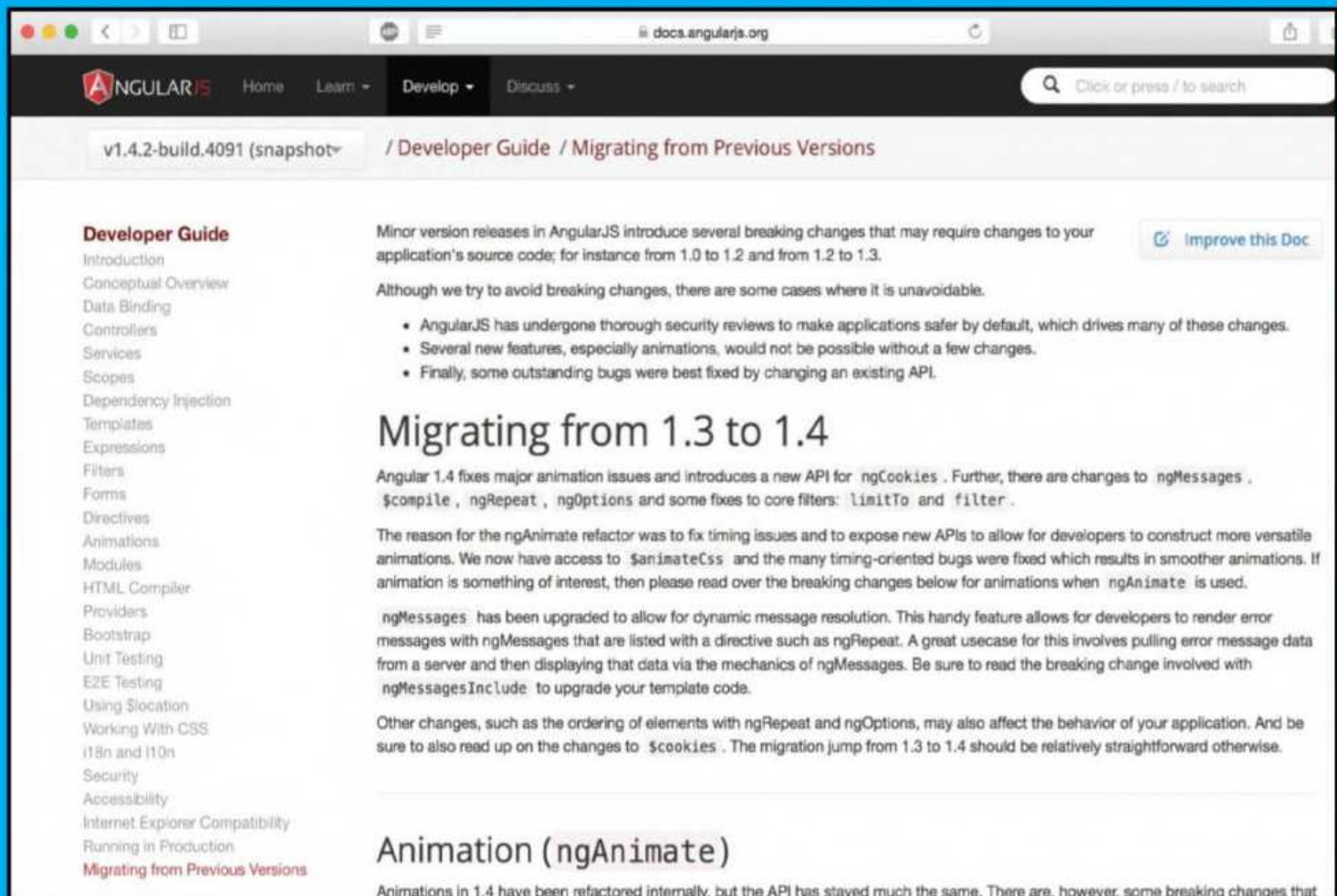
<p class="error" data-ng-
message="required">Please enter your
telephone number.
</p>
```

```
<p class="error" data-ng-message="tel,
pattern">Please enter a valid telephone
number.
</p>
</div>
```

16. Postcode input markup

You wouldn't think it but postcodes are actually quite hard to get right because there are so many variations. Nonetheless, we're going to create a custom directive to try and validate one.

```
<div class="row">
<label for="postcode">Postcode:</label>
<input id="postcode" name="postcode"
data-ng-model="fields.postcode" postcode
required />
</div>
```



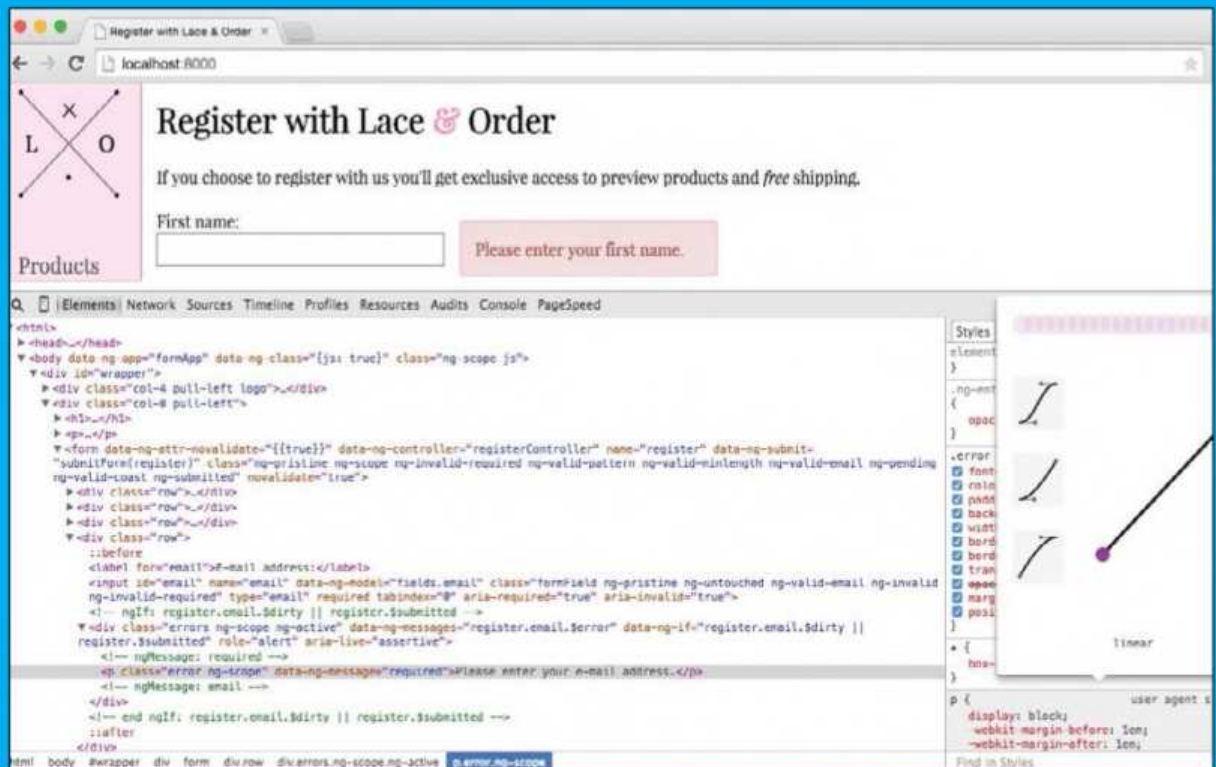
What's new in Angular 1.4?

Angular 1.3 introduced ngMessages as an experimental module. This meant that developers were free to use it but breaking changes could be made to the API. The main change in 1.4 is that ngMessagesInclude was removed as an attribute on ngMessages. Instead it's used on a ngMessage directive, like we've used in this tutorial. Angular 1.4 also introduced ng-message-exp and when-exp to evaluate expressions dynamically. Upgrading from 1.3 to 1.4 should be straightforward for most projects and you can find detailed migration information at docs.angularjs.org/guide/migration.

Breathe life into ngMessages

You can add a sense of momentum to your messages with ngAnimate. We've already installed and included this module. When ng-messages is shown, a class of ng-enter and ng-active is added - the former when it's being inserted and the latter when it's active. Our message boxes all have a class of 'error' so we'll target this and change the opacity. The CSS transition property will then smoothly animate the box in and then out when it's removed. You can use any sort of animation to be more creative here.

```
.error {
  transition:
    0.25s linear;
  opacity: 0;
}
.ng-enter .error,
.ng-active .error {
  opacity: 1;
}
```



17. Postcode directive

Create a new file under directives called 'postcode.js'. By requiring ngModel it's passed as the fourth argument to the link function so that we can extend it. Remember to add it as a dependency of the registration module.

```
angular.module('postcode.directive', [])
.directive('postcode', function() {
  return {
    require: 'ngModel',
    scope: {
      postcode: '='
    },
    link: function (scope, elm, attrs, ctrl) {}
  };
})
```

18. Regular expression test

Within the link function we'll write the postcode validator. logic The long, nasty looking regular expression validates most UK postcodes. To register a new validator to the model that this directive is on we add to \$validators. Now close spaces and see if the string passes REG_EXP.

```
var REG_EXP = new RegExp(/^[([g][i][r][0][a]
[a])$|^([a-pr-uwyz]{1}
([0][1-9]\d?)|([a-pr-uwyz]{1}[a-hk-y]{1}
([0][1-9]\d?)|([a-pr-uwyz]{1}[1-9]
[a-hjkps-uw]{1})|([a-pr-uwyz]{1}[a-hk-y]{1}
[1-9][a-z]{1}))(\d[abdhjlnp-uw-z]
{2})?)$/i);
ctrl.$validators.postcode = function (value) {
  if (typeof value === 'string') {
    value = value.replace(' ', '');
    return REG_EXP.test(value);
  }
};
```

19. Include message template

The custom validator is then added as another property to the \$error object if it returns false. This means that it can be used within ng-message. We're also going to use a template to fill in common error messages. This is included with ng-messages-include.

```
<div class="errors" data-ng-
messages="register.postcode.$error">
```

```
data-ng-if="register.postcode.$dirty ||
register.$submitted" role="alert">
<p class="error" data-ng-
messages="postcode">Please enter a valid
postcode.
</p>
<div data-ng-messages-include="default-
error-messages">
</div>
</div>
```

20. Custom message template

Ng-messages-include looks for an element's ID on the page. Angular recommends including templates as a script tag with a custom type so that it doesn't end up being parsed by the browser. The downside of this approach is the generic wording although this could be paired with dynamic text.

```
<script type="text/ng-template" id="default-
error-messages">
<p class="error" data-ng-
message="required">This field is required.
</p>
<p class="error" data-ng-
message="minlength">This field is too short.
</p>
<p class="error" data-ng-
message="maxlength">This field is too long.
</p>
</script>
```

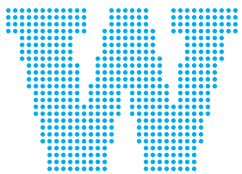
“ This means that it can be used within ng-message. We're also going to use a template to fill in error messages ”



Create an interactive 360° scene in WebGL

Make panoramic explorable environments with the three.js

WebGL library **tools | tech | trends** Brackets, three.js



WebGL is growing in popularity as a rich graphics-rendering platform for the browser. It boasts real-time 3D graphics that are hardware

accelerated by the graphics card and is available on all major browsers, across all major platforms. Three.js is the

original library for creating 3D content for the web and it is consistently updated. In this tutorial we will create a 360-degree panorama scene. To get around lower-specifications problems, prerender the scene in a 3D package and use a spherical map to add this to the sphere.

Once mapped back onto a sphere, the spherical map will look like the real scene. There are cameras and

lenses available that create this kind of spherical image for real-world photography. But we want users to be able to interact so we will create hotspots that show a tool tip when the user rolls over them. The tool tip will be a regular HTML div tag, styled up with CSS. JavaScript will detect the hotspot we are over and display the right content, switching the tool tip off when not over a hotspot.

1. Start the project

To begin, open up the start folder in Brackets or place it in your local web server folder and edit the index.html page in your code editor. Then, in the body section add the code shown below to start the project. We are going to have a tool tip called 'info'. Place a logo in the top-left corner and then render our WebGL scene into the container element.

```
<div id="info" class="hidden">Hello
World!</div>
<div id="logo"></div>
<div id="container"></div>
```

2. Style these elements

In the head section of the document, add the style tag as shown in the style rules. For the body we are making the background black, turning the margins off and setting any overflow content to hidden. The logo is being positioned over the top of the 3D content in the top-left of the screen.

```
body {
background-color: #000000;
margin: 0px;
overflow: hidden;
}
#logo{
position: absolute;
top: 0px; left: 0px;
width: 274px;
}
```

3. Style the tool tip

Let's style the tool tip up to just sit in the top-left of the screen for now, later we'll update this as the mouse moves so that it will end up following the mouse. We'll give it a slightly transparent, black background with a very small border radius on the corners. White text will finish off the tool tip.

```
#info {
position: absolute;
top: 0px; left: 0px;
}
background: rgba(0, 0, 0, 0.7);
border-radius: 2px;
color: #fff;
padding: 5px;
font-family: Arial;
font-size: 18px;
text-align: center;
}
```

4. Hide and show the tool tip

We're creating two additional classes. These classes will turn the visibility of the tool tip on and off. By default we've set it to hidden here, but when we move our mouse over a 3D model we'll bring the tool tip back to show the user what the area of interest for them to be clicking on is. This will give us a tool tip that shows itself at the appropriate time.

```
.hidden{ visibility: hidden; }
.visible{ visibility: visible; }
</style>
```

5. Start the code

Before the final body tag add the code shown below. Here we are linking to the three.js library then starting the variables that we will use in the project. All our remaining code in the steps to follow will be placed before the closing script tag. The variables are tracking the mouse position and looking after firing a ray into the scene from the position of the mouse.

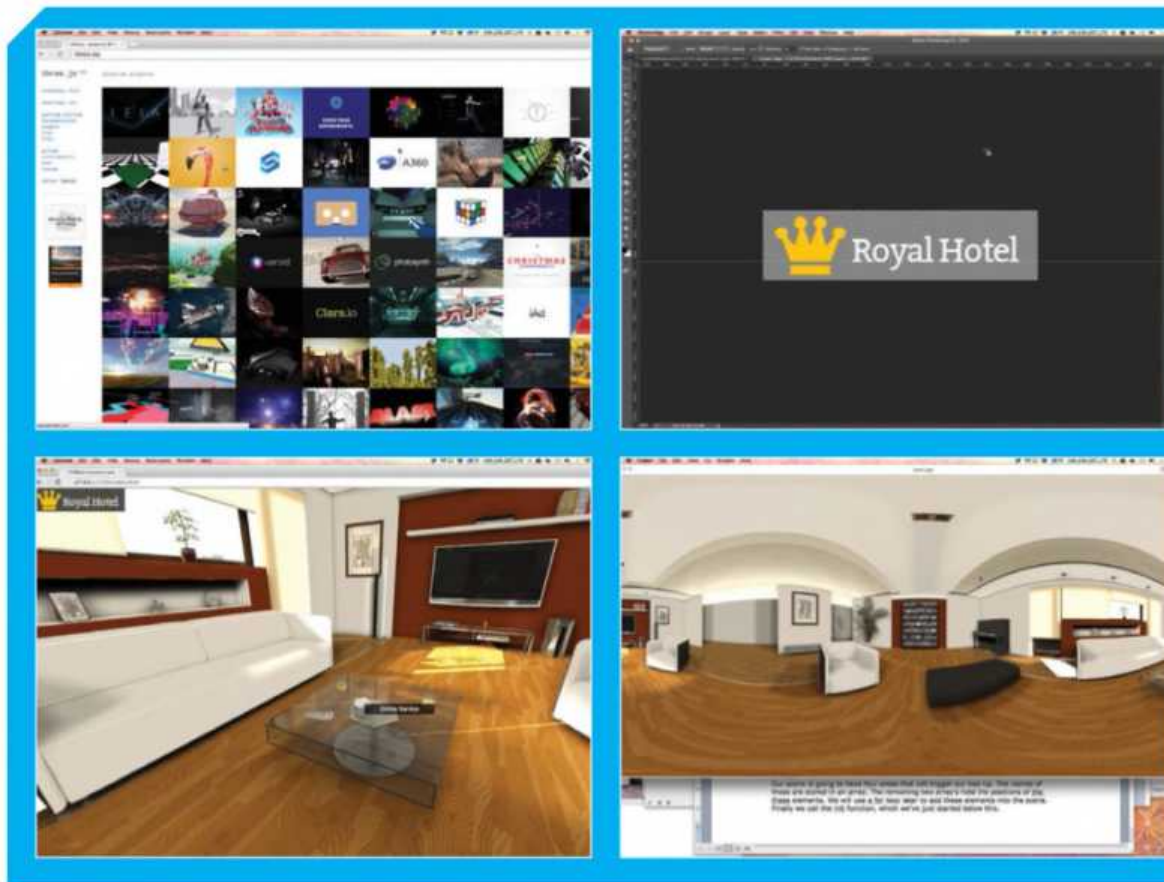
```
<script src="js/three.min.js"></script>
<script>
var camera, scene, renderer;
var isUserInteracting = false,
onMouseDownMouseX = 0, onMouseDownMouseY =
0, lon = 0, onMouseDownLon = 0, lat = 0,
onMouseDownLat = 0, phi = 0, theta = 0;
var mouse = new THREE.Vector2(), raycaster,
INTERSECTED, hover = false, info;
</script>
```

6. Area of interest

Our scene is going to have four areas that will trigger our tool tip and the names of these will be stored in an array. The remaining two arrays will hold the positions of these elements. We will use a 'for' loop later so that these elements are added into the scene. Finally we call the init function.

7. Create the scene

Here we will move on to declaring some more variables that we will use in the init function. We also cache a reference to the info and container DOM element. We will then create a camera to look into our scene and then set



<Top left>

- The WebGL library that we are using is in both the start and finished folder but you can find its project home at <http://threejs.org>, where more examples and documentation can be found

<Top right>

- In Step 1, we are displaying a hotel logo in the corner. If a hotel had executive suites they would probably want to show off the rooms in more detail using an interactive scene

<Bottom left>

- Here we have added classes that with the help of some clever JavaScript turn the tool tip on and off by toggling the visibility property in CSS

<Bottom right>

- The flat image that we are loading and wrapping around the inside of a sphere can be seen here with the screenshot. We load it and use it as a texture on the sphere geometry

this camera to look at the centre of the scene. Finally we will create a new scene to add all of our content to.

```
var container, mesh;
info = document.getElementById( 'info' );
container = document.getElementById(
'container' );
camera = new THREE.PerspectiveCamera( 75,
window.innerWidth / window.innerHeight, 1,
1100 );
camera.target = new THREE.Vector3( 0, 0, 0
);
scene = new THREE.Scene();
```

8. Make the room

Create a large sphere and turn it inside out by scaling it to -1 on the x axis. Then we create a material for this and load in the texture to display. Next add the geometry and the material together to create our 3D model, which is then added to the scene so that it is visible to the camera.

```
var geometry = new THREE.SphereGeometry(
500, 60, 40 );
geometry.applyMatrix( new THREE.Matrix4().
makeScale( -1, 1, 1 ) );
var material = new THREE.MeshBasicMaterial(
{ map: THREE.ImageUtils.loadTexture( 'img/
room. jpg' ) } );
```

```
mesh = new THREE.Mesh( geometry, material
);
scene.add( mesh );
```

9. Create another sphere

Add more spheres to the edge of our first sphere. These will give us hotspots that we can hover the mouse over and get information. Then reuse the variable's geometry and material to create our new sized spheres. Now we use a 'for' loop to loop through the names in our array. Each sphere is given a unique name and made invisible so we get invisible hotspots.

10. Position each sphere

Use the array of pos1 and pos2 to position our spheres at different places over the edge of our first sphere, which contains the room image. We are using mathematical functions of sin and cos to position the hotspots out from the centre of the scene and onto the room image sphere.

```
var phi2 = Math.acos( pos1[i] );
var theta2 = Math.sqrt( Math.PI ) *
pos2[i];
mesh.position.x = 530 * Math.cos( theta2 )
* Math.sin( phi2 );
mesh.position.y = 530 * Math.sin( theta2 )
* Math.sin( phi2 );
mesh.position.z = 530 * Math.cos( phi2
);
```

Radians not angles

Computers take angular data as radians not angles - 360° is equal to $\text{PI} \times 2$ as radians, so a full turn is roughly 3.28 as radians. It can be quite confusing if you are new to this!

```
};
```

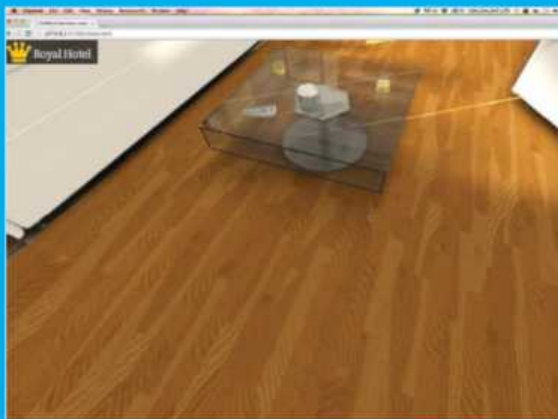
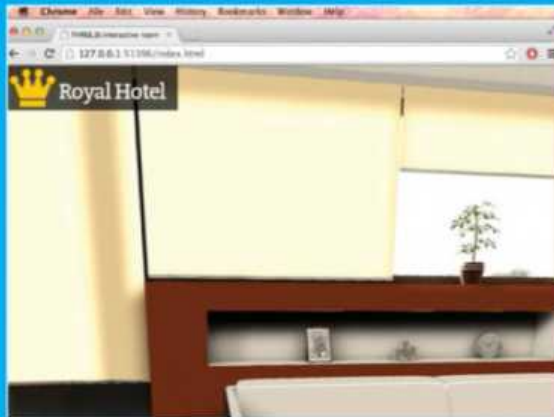
11. Render the scene

Now set up WebGL to render our 3D scene. The size of this is set to the inside dimensions of our browser. The container div gets this renderer added to it. Next we set an important variable, which is the raycaster. This fires an invisible ray into the scene from the position of the mouse and tells us which models are under it.

12. Event Listeners

We need to control what happens when the mouse moves or when the mouse is pressed and released. For this we add event listeners, which listen to these user interactions. We will add one more listener, this is the resize event and handles what happens when the browser window is resized.

```
document.addEventListener( 'mousedown',
onDocumentMouseDown, false );
```



◀Top left▶

- In this step we add spheres as hotspots into the scene. In this screenshot we have left the visibility of the models on so that you can see the hotspots, however we turn them off and use them as hit areas only in the final project

◀Top right▶

- We have enabled the browser to be resized by updating the render size and the aspect ratio of the camera as the window changes. This also lets our 3D scene be responsive

◀Bottom left▶

- If the user clicks in the scene and drags with their mouse they can change the angle of viewing the scene. This enables a full 360-degree view of the room

◀Bottom right▶

- When the user puts their mouse over a hotspot, a ray is sent into the scene bringing back an array of the objects under the mouse. Depending which object it is, we bring up the appropriate tool tip

Array variables

Array variables are best to think of as lists of useful information. We usually use a 'for' loop to iterate through all of the elements in the list and do something with them.

```
document.addEventListener( 'mousemove',
onDocumentMouseMove, false );
document.addEventListener( 'mouseup',
onDocumentMouseUp, false );
window.addEventListener( 'resize',
onWindowResize, false );
animate();
}
```

13. Resize the window

If the browser window is resized by the user, we listen to that event and then we update the way our scene displays because of that. Grab the inner width and height of the browser window and then update the camera's aspect ratio. Finally, change the size of our renderer as well.

```
function onWindowResize() {
    camera.aspect = window.innerWidth /
    window.innerHeight;
```

```
camera.updateProjectionMatrix();
renderer.setSize( window.innerWidth,
window.innerHeight );
}
```

14. Press the mouse

When the mouse is pressed there are a few things that we want to do. The first is to stop the default event from propagating. We set the `isUserInteracting` flag to true and record the mouse's x and y position ready to update our display in the browser as the mouse drags the scene around.

```
function onDocumentMouseDown( event ) {
    event.preventDefault();
    isUserInteracting = true;
    onPointerDownPointerX = event.clientX;
    onPointerDownPointerY = event.clientY;

    onPointerDownLon = lon;
    onPointerDownLat = lat;
}
```

15. Move the scene

When the mouse moves around we want to detect if the user has pressed the mouse, if they have then the `isUserInteracting` variable will be true. Update the longitude and latitude variables (these will be based on

the mouse position) and then we can move the scene. We also set the tool tip to follow the mouse.

```
function onDocumentMouseMove( event ) {
    if ( isUserInteracting === true ) {
        lon = ( onPointerDownPointerX - event.
        clientX ) * 0.1 + onPointerDownLon;
        lat = ( event.clientY -
        onPointerDownPointerY ) * 0.1 +
        onPointerDownLat;
    }
    mouse.x = ( event.clientX / window.
    innrWidth ) * 2 - 1;
    mouse.y = - ( event.clientY / window.
    innerHeight ) * 2 + 1;
    info.style.top = event.clientY - 50
    + "px";
    info.style.left = event.clientX + 20 +
    "px";
}
```

16. Mouse released

When the user releases the mouse we tell the `isUserInteracting` variable to be false so that the scene doesn't update and change. At this point we run our `animate` function this gets called 60 times per second and runs the `update` function which updates the scene.

<Below>

- Now we target all the p tags within our columns and give them default styles as well as 15 pixel padding to make a gutter



Creating the room

The room image is created in a very specific way in order for it to render correctly, by wrapping it around a sphere. From the start folder, view the img folder and you will see the image room.jpg inside. Open this image in Photoshop to view the flat image. The image was rendered by Cinema 4D from a model of a 3D scene. We use this flat image in WebGL when we want a scene to look better than what the 3D renderer can actually create by loading the model.

This image alone took 22 minutes to render, which isn't exactly real-time. To get this image we add a sphere to the centre of our scene and set its reflections to full. We then render the texture from the sphere and as it is receiving the full reflections of the room, we end up with a spherical map of the room that we can then use in WebGL.

```
function onDocumentMouseUp( event ) {
  isUserInteracting = false;
}
function animate() {
  requestAnimationFrame( animate );
  update();
}
```

17. Update the display

Let's create the update function, now. Check if the user is hovering over a hotspot or if they're interacting with the scene. If it's the latter, update the scene, otherwise set the scene to slowly rotate.

```
function update() {
  if ( hover == true || isUserInteracting == true ) {
    lon = lon;
  } else {
    lon += 0.05;
  }
  lat = Math.max( - 85, Math.min( 85, lat ) );
  phi = THREE.Math.degToRad( 90 - lat );
  theta = THREE.Math.degToRad( lon );
```

18. Update the camera target

Change the camera target so that it rotates around the circumference of the sphere with the room image mapped onto it. This means that the camera stays still in the centre while the target rotates around the centre point. If the user drags the scene, these values change and the camera looks at the new position.

```
camera.target.x = 500 * Math.sin( phi ) *
Math.cos( theta );
camera.target.y = 500 * Math.cos( phi );
camera.target.z = 500 * Math.sin( phi ) *
Math.sin( theta );
camera.lookAt( camera.target );
```

19. Shoot rays

Next, shoot a ray into the scene from the the mouse's x and y position from the point of view of the camera. Then check to see if the ray intersects any objects in the scene. This will make our 'intersects' array contain a list of all models under the mouse.

```
var vector = new THREE.Vector3( mouse.x,
mouse.y, 1 ).unproject( camera );
raycaster.set( camera.position, vector.sub(
camera.position ).normalize() );
var intersects = raycaster.
intersectObjects( scene.children );
if ( intersects.length > 0 ) {
```

20. Under the mouse

We named all of our hotspots earlier in Steps 6 and 9. Here we are checking to see if the hotspot named 'ent' is below the mouse. If it is, then we turn the visibility of the tool tip on and set the text of the tool tip to give relevant information. We also set our hover variable to true to stop the scene automatically spinning.

```
if ( INTERSECTED !== intersects[ 0 ].object
) {
  INTERSECTED = intersects[ 0 ].object;
```

```
if ( INTERSECTED.name == "ent"){
  hover = true;
  info.innerHTML = "Entertainment System";
  info.classList.remove('hidden');
  info.classList.add('visible');
}
```

21. Check other objects

Using the same principle as in the previous step we detect if the model hotspot under our mouse is the piano or the table. For each of these we set the text in the hotspot to display the correct name and set the hotspot to be visible. This is repetitive but necessary.

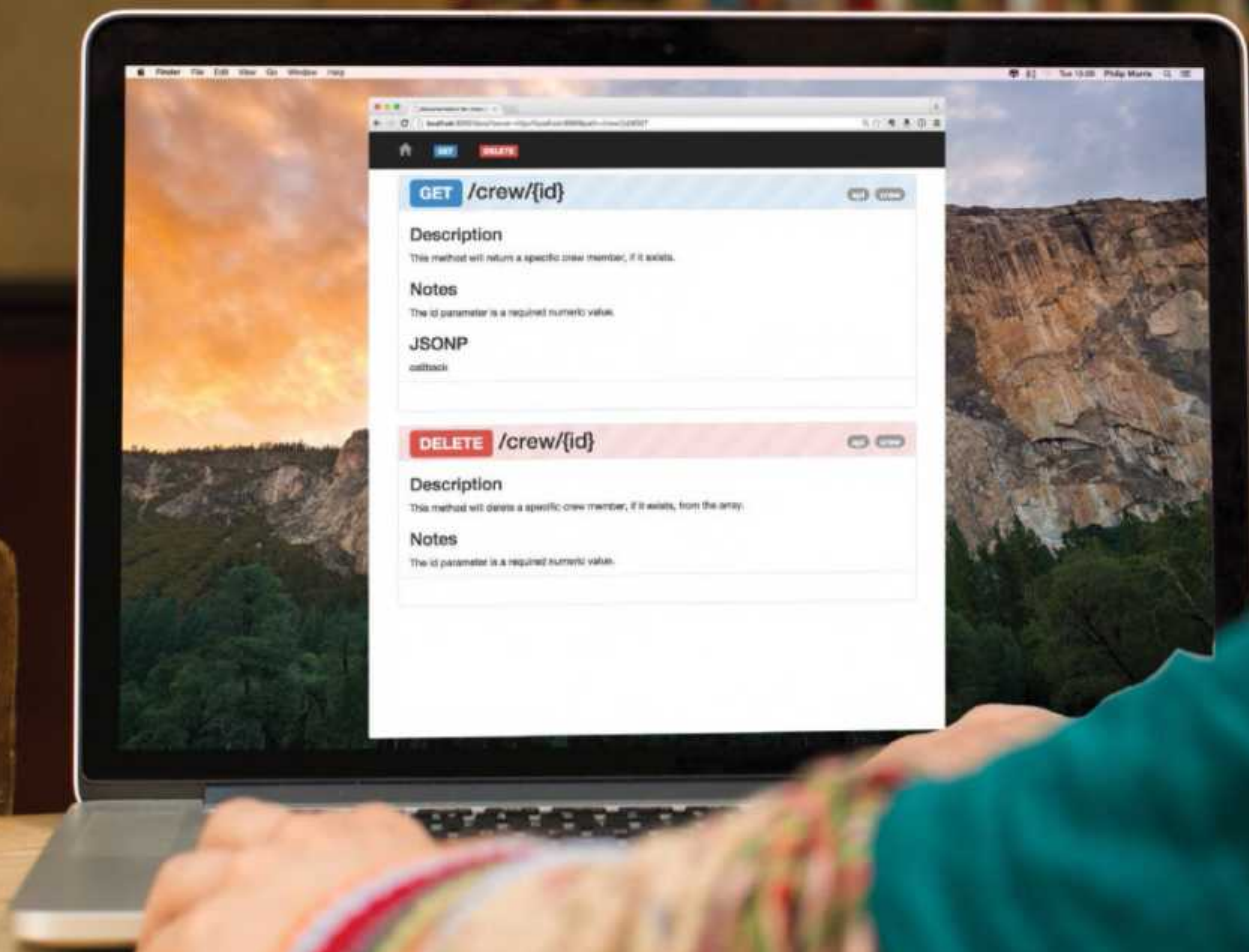
```
else if ( INTERSECTED.name == "piano"){
  hover = true;
  info.innerHTML = "Piano";
  info.classList.remove('hidden');
  info.classList.add('visible');
} else if ( INTERSECTED.name == "table"){
  hover = true;
  info.innerHTML = "Drinks Service";
  info.classList.remove('hidden');
  info.classList.add('visible');
}
```

22. Finish off the project

Check if the library is under the mouse and change the tool tip for that. Our final else statement just detects none of the hotspots and therefore turns the tool tip off. The very final part is to update the display on the screen. Save the document and check it in your browser from a web server.

Build your own API with the Hapi.js framework

Develop a feature-rich API using Hapi with reusable application logic and minimal code





APIs are everywhere, just look around the web. Most services have an API that you can consume or interact with in some way. They offer extensibility, allowing developers and

consumers to build enriched detailed applications, and they also combine multiple API sources if needed to develop something that is truly unique or specific to requirements.

They also offer developers the structural work for building applications for multiple devices, just create your underlying API and consume it in the browser or on mobile devices. They benefit from being lightweight, portable and easily maintainable.

There are a vast number of open source API solutions available for a large number of languages. In this tutorial we will look at Hapi.js, a JavaScript solution offered by the team at Walmart Labs that helps you build powerful APIs and applications with minimal code.

As we use Node underneath we will begin by installing the relevant required packages to generate our library before we create specific API routes to manage fetching, updating and deleting records from our data structure. We'll also look into the importance and ease of creating documentation using the framework.

If you have not explored creating an API for your web services yet, Hapi.js may be the very thing you've been waiting for to help you get started.

1. Install required packages

First we need to generate a new Node package file for our project and install the required Hapi module. Using

the command line, navigate to your desired project location and initialise a new package.json file and accept as many default values as you need to. Once complete, install the API module and save it as a dependency into the package.json file.

```
npm init
npm install hapi --save
```

2. Create the server

Create a new file called 'server.js'. This will hold the core code for instantiating the server and dealing with routes. Require the API library and assign it to a variable of 'hapi'. Finally, create a new server instance and assign that to a new variable called 'server'.

```
var hapi = require('hapi');
var server = new hapi.Server();
```

3. Define the connection

With the server created we can now define the connection details. Using the underlying API we can implement the server.connection() method to define the connection details. Set the host and TCP port details as optional arguments. The port can be passed as a command-line argument or will default to 8080.

```
server.connection({
  host: 'localhost',
  port: Number(process.argv[2] || 8080)
});
server.start(function () {
  console.log('Server running at:', server.
```

```
info.uri);
});
```

4. Run the server

You can simply run 'node server.js' which will fire up the server, but this will restrict your workflow if you make any changes. Install the supervisor package as a global module and then just run Node from there. It will reload the process whenever you update the file and you will then see the console output and an error JSON response in your browser.

```
node server.js
npm install supervisor -g
supervisor server.js
```

5. Define routes

The server needs to have routes defined to handle the request and output accordingly. Create a new route, providing the URL path and the HTTP method associated with the route. The handler option will call a function which will manage the response to the browser and output a simple string message.

Multiple methods

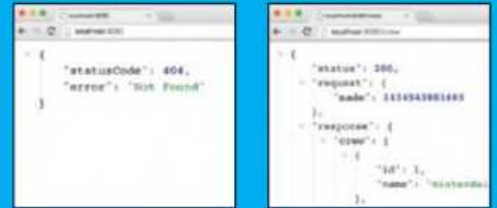
You can declare multiple methods in your route definition as an array like so: ['PUT', 'POST']. Hitting this endpoint with either of these methods will return the same response.

```
hapi -- npm init -- npm -- node

Press ^C at any time to quit.
name: (hapi)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
license: (ISC)
About to write to /Users/monkehworks/Desktop/hapi/package.json:

{
  "name": "hapi",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Matt Gifford <matt@monkehworks.com> (http://monkehworks.com/)",
  "license": "ISC"
}

Is this ok? (yes) |
```



<Left>

- Simplify your tasks and let npm create the package.json file for you using sensible default values

<Top left>

- The initial server start returns an error JSON response as we have no routes defined for the application

<Top right>

- The GET request to the /crew route returns the JSON structure of data as expected

Development

```
server.route({path: '/', method: 'GET',
  handler: rootHandler});
function rootHandler(request, reply){
  reply('API Hapi-ness');
};
```

6. Additional route definition

Create another route for the server with the path `/crew`, which will call a function called `getCrew` when accessed using the HTTP GET method. Here we have sent the handler through as a parameter in the route options config object, which can be used to set a number of options to fine-tune each route.

```
server.route({
  method: 'GET',
  path: '/crew',
  config: {
    handler: getCrew
  }
});
```

The documentation

The Hapi library helps to make creating powerful but lightweight APIs a simple task, and it has an extensive API itself which you can tap into. Check it out here: hapijs.com/api.

7. Build the response

The `getCrew` method has access to both the request and reply objects. We'll use `reply` to send the response back to the browser. The generated response will provide a JSON object containing a status and the returned data, which is obtained from an array of objects in the code.

```
function getCrew(request, reply) {
  reply({
    status: 200,
    request: {
      made: request.info.received
    },
    response: {
      crew: crewDetails
    }
  });
}
```

8. Sample data structure

Our app needs data to manage and return, so we'll use a manually created array of objects, contained within the JavaScript file. Whilst this suits our purpose for a sample API, you can connect your API to use any available data source with an associated Node module for connectivity.

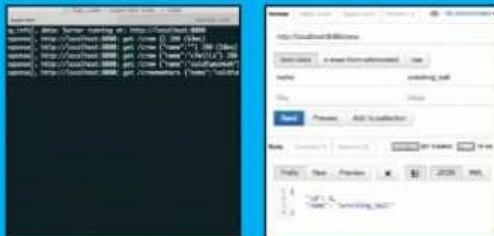
```
var crewDetails = [
  {
```

```
    id: 1,
    name: 'misterdai'
  },
  {
    id: 2,
    name: 'fynd'
  }
];
```

9. Access URL parameters

We can use URL parameter values in our routes to filter data. Adjust the `getCrew` handler method to respond with a new method (`findCrewMember`) if the `name` key is found within the request query object, passing it through as an argument to the function.

```
var crewResponse = crewDetails;
if (request.query.name) crewResponse =
  findCrewMember(request.query.name);
reply({
  status: 200,
  request: {
    made: request.info.received,
    params: request.query
  },
  response: {
    count: crewResponse.length,
    crew: crewResponse
  }
});
```



<Top left>

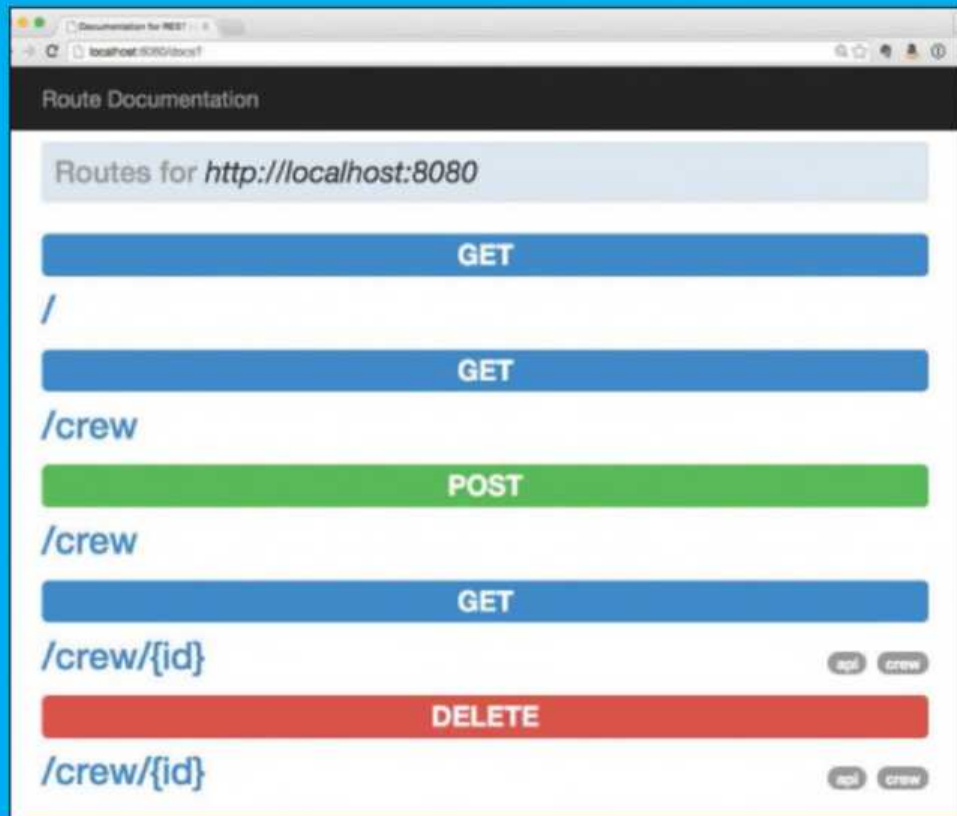
- Using the good-console plugin really helps to structure request logs and to see what parameters are being passed through

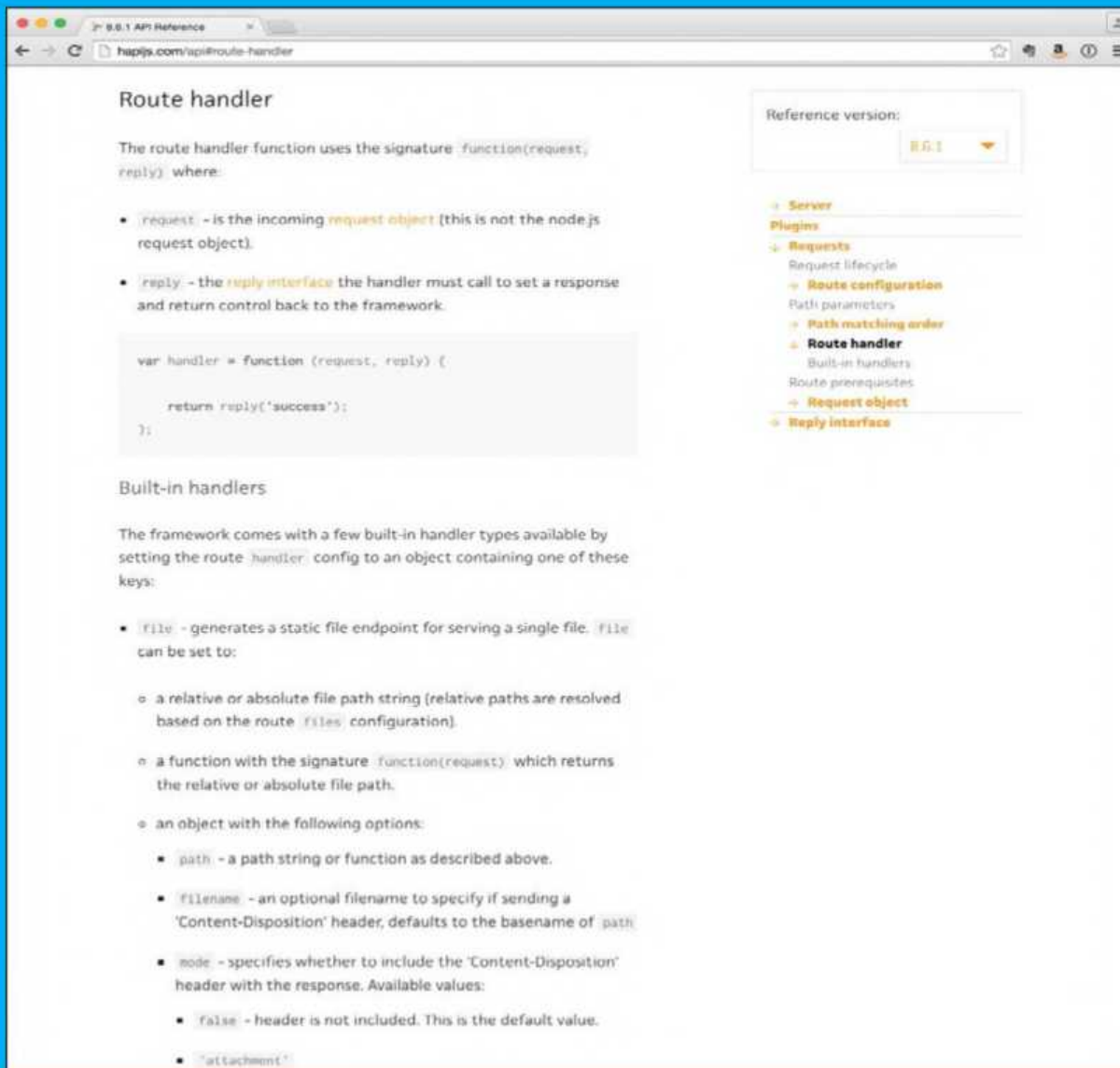
<Top right>

- Use an app like Postman to interact with your local API. Here we send the POST request to add a new member

<Right>

- Self-documenting applications and plugins such as Lout can really improve your development workflow and automation processes





Built-in handlers

Building up a `server.route` definition means associating it with a handler of some kind, typically a function to manage data manipulation and output as we have seen in the full tutorial code. Hapi.js includes a number of built-in handlers to offer some alternate but important options as standard.

Your application may require some static files for display as well as any API JSON responses from certain endpoints, for example the default `/` route may display an index.html page. Hapi.js enables you to use the file handler to cater for this, as well as for any directory and proxy handlers. The detailed documentation has all of the configuration options that you need, head over to hapijs.com/api#route-handler to find more information.

10. Filter data

The new `findCrewMember` function accepts the provided name value as the method argument and uses it to filter the array, which is then returned directly to the browser in the expected JSON response format and structure. This has additional values to also show the request URL parameters.

```
function findCrewMember(name) {
  return crewDetails.filter(function(member) {
    return member.name.toLowerCase() === name.toLowerCase();
  });
}
```

11. Validation and logging

As we're now dealing with extra parameters, now would be a good time to employ some additional tools. Run the

“ The API has a `server.register` method available for this purpose which can take an array or plugin objects, each of which handles its own option management ”

npm command to install three packages: `Joi`, `good` and `good-console`, saving them to the `package.json` file for future use. Once done, add the two `require` statements to the JavaScript file.

```
// CLI
npm install joi good good-console --save
// JS file
var joi = require('joi');
```

```
var good = require('good');
```

12. Register the plugin

The server needs to be made aware of some plugins such as the `good` reporter we will use. The API has a `server.register` method available for this purpose which can take an array or plugin objects, each of which handles its own option management to send values to the plugin for use.

```
server.register([{\n  register: good,\n  options: {\n    reporters: [{\n      reporter: require('good-console'),\n      events: { response: '*', log: '*' }\n    }]\n  },\n  function (err) {\n    ...\n  }]);
```

13. Plugin error handling

It's always wise to be prudent when registering any plugins to the server and catch any errors that may arise. Use the register method's callback option to detect any errors and throw an exception if needed. Move the server.start method into the callback following complete successful registrations.

```
if (err) throw err;\nserver.start(function () {\n  server.log('info', 'Server running at: ' +\n    server.info.uri);
```

Hapi plugins

Whether you need authentication, security or localisation features in your app, Hapi.js can be extended with a number of plugins at hapijs.com/plugins#Authentication.

```
});
```

14. Route input validation

Here we have passed a validate.query object to the route config object to tell Hapi that we want the query parameter values specified to be validated. The name has been restricted to meet particular minimum and maximum length requirements.

```
config: {\n  handler: getCrew,\n  validate: {\n    query: {\n      name: joi.string().min(4).max(25),\n      callback: joi.string()\n    }\n  },\n  jsonp: 'callback'\n}
```

15. Route with parameters

Create a new route in the file that enables us to find a specific crew member using a parameter as part of the route, denoted here using the parenthesis. The config block will call the handler function and we also include optional meta information with descriptive details.

```
server.route({\n  method: 'GET',\n  path: '/crew/{id}',\n  config: {\n    handler: getMember,
```

```
    jsonp: 'callback',\n    description: 'This method will return a\n    specific crew member, if it exists.',\n    notes: 'The id parameter is a required\n    numeric value.',\n    tags: ['api', 'crew']\n  }\n});
```

16. Find a member

The getMember handler function will be able to filter the static data array using the numeric value sent through the request parameters as the id that is taken from the route. Once again, the JSON response is generated and sent back to the client. If no member exists, an empty object is returned in the block.

```
function getMember(request, reply) {\n  var member = crewDetails.\n  filter(function(member) {\n    return member.id === parseInt(request.\n    params.id);\n  }).pop();\n  if (typeof(member) !== 'object') member =\n  {};\n  reply({\n    status: 200,\n    request: {\n      made: request.info.received,\n      params: request.params\n    },\n    response: {\n
```



<Top left>

- Filtering out records using URL query parameters is easily achieved. Here we also return the param values in the response

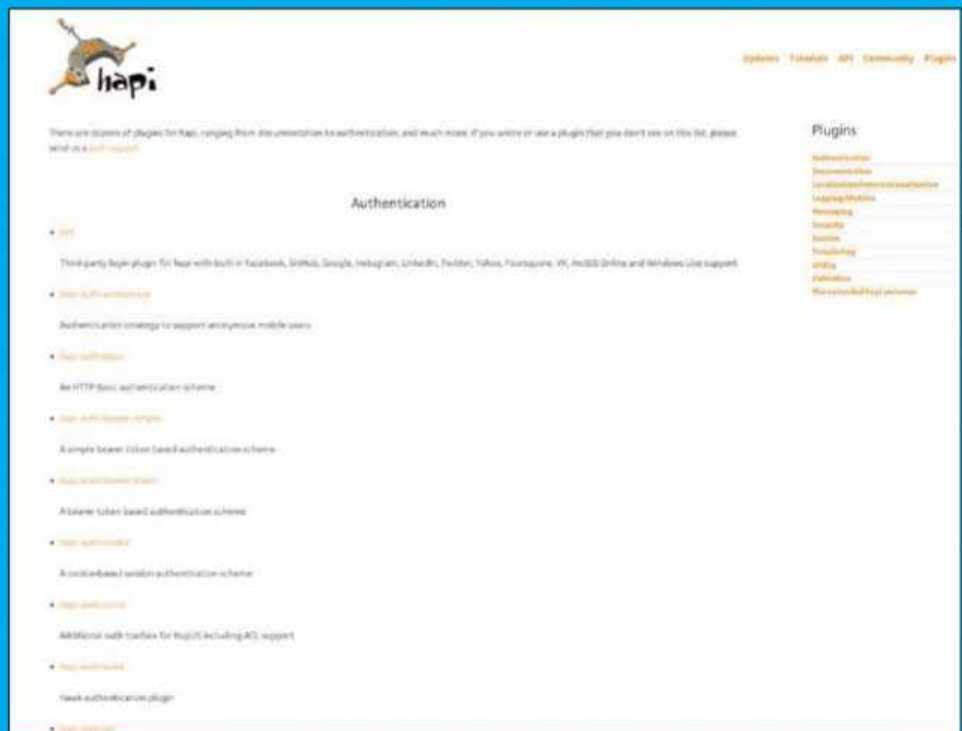


<Top right>

- The 400 bad request error will occur if our name value is below the minimum of 4 characters

<Right>

- Authentication plugins can be downloaded from the Hapi site. Bell is a third-party plugin that ships with support for Facebook, Twitter, GitHub and so on



```

crew_member: member
}
});
}

```

17. Create a post route

Create a new route with a POST method that will accept a payload containing a name key value pair. Use Joi to set validation requirements on the data input. The handler method will then take the payload and update the static array.

```

server.route({
  method: 'POST',
  path: '/crew',
  config: {
    handler: addMember,
    validate: {
      payload: {
        name: joi.string().required().min(3)
      }
    }
  }
});

```

18. Add data to array

Following a POST request to the new route the addMember function will be called. This will generate a new member object with an incremented id value, taken from the current array length plus 1. The correct 201 status code is returned to the user following the new record creation.

```

function addMember(request, reply)
{
  var member = {
    id: crewDetails[crewDetails.length - 1].id + 1,
    name: request.payload.name
  };
  crewDetails.push(member);
  reply(member).code(201).header('Location',
    '/crew/' + member.id);
}

```

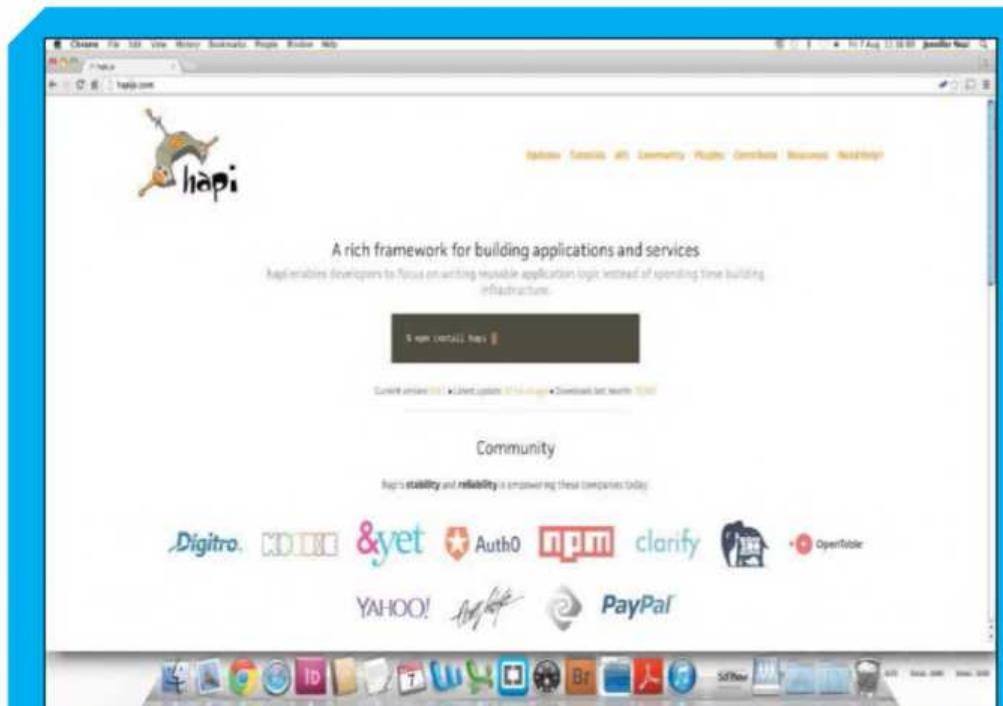
19. Delete member handler

Create a new route definition similar to the getMember route which accepts a DELETE HTTP method request. This will call the deleteMember method to remove the specific object from the array using the id value sent as the request parameter.

```

function deleteMember(request, reply) {
  var member;
  for ( var i = 0; i < crewDetails.length; i++) {
    if ( crewDetails[i].id === parseInt(request.params.id) )
    {

```



<Above>

• You can download everything you need from the Hapi website.

Documentation is crucially important for any application, and this is certainly true for an API that could be opened and available for consumption

```

member = crewDetails[i];
crewDetails.splice(i, 1);
reply(member).code(200).header('Location',
  '/crew/' + member.id);
break;
}
}

```

20. Self-document your app

Documentation is crucially important for any application, and this is certainly true for an API that could be opened and available for consumption by developers and organisations needing to understand how it all works. We will install a new Node module called Lout to help us out with this process, and we will be saving it to the package.json file in the process.

```

npm install lout --save
<p class="error" data-ng-
message="maxlength">
This field is too long.
</p>

```

</script>

21. Register Lout

As it is an additional plugin we need to register Lout with the server, as we have already done with the good-console module. Revise the array of plugin objects and add a new one that registers the Lout module.

```

server.register
([
  { register: require('lout') },
  {
    register: good,
    ...

```

22. Documentation generated

With the plugin registered and any meta information (description, tags and notes) defined for each of our routes, simply navigating in the browser to /docs will display the Lout documentation. The styles can be overridden to suit your requirements. Selecting a specific method will display all associated meta information.

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Manage your project's dependencies with jspm

Master frictionless package management and learn how to manage and bundle your projects **tools | tech | trends** jspm, npm, JavaScript, Terminal.app



For the past few years we've found ourselves jumping through hoops to bring a robust module system to JavaScript. Tools like RequireJS and Node paved the way in popularising modules

and approaches like AMD, UMD, and CommonJS. ECMAScript 6 (the next version of JavaScript)

promises to unify all of these approaches with native modules.

Jspm provides a way to use these technologies today as well as all of the features you'd expect from a package manager. Jspm uses SystemJS to load ES6, AMD and CommonJS modules, this then works in tandem with Traceur or Babel to compile this ES6 code to ES5, which all modern browsers can run (as well as IE 9+). Jspm is a

command-line tool so some prior experience with using Terminal.app or cmd.exe would be beneficial. Don't worry if you haven't written any ES6 prior to this, we'll explain as we go.

We're going to use Google Maps to create a map, load some data, and plot some markers on the map. Through this you'll learn some real-world examples of how ES6 modules can help decouple your code.

1. Install jspm

Jspm is installed via npm, this is a little strange as you're using a package manager to install a package manager. We'll head to nodejs.org to install it and do this globally so that it's available throughout the system.

```
$ npm install -g jspm
```

2. Set up the project

With jspm installed we'll create a new folder for our demo and navigate into it. The jspm CLI has an init command which asks a few questions about the structure of your app, you can likely just hit Enter to select the default. We've chosen to use the Babel transpiler over Traceur.

```
$ mkdir jspm-demo && cd jspm-demo
$ jspm init
```

3. Download data

The init process will create a few files for you, including package.json and config.js. You should rarely have to edit these directly as jspm does most of the maintenance of these files for you. We're going to be using this data source from http://bournemouthdata.io/data/buildings_listed_buildings.csv which you'll need to download a local copy of under a folder called 'data'.

4. index.html

Below is a reduced version of our index.html file. Jspm uses SystemJS to do the actual in-browser module loading. We then point the browser to our configuration

file which passes important information to SystemJS like where to look for our modules and what to map the names to. It is this mechanism which enables shorter names than 'github/jquery/jquery'.

```
<!doctype html>
<title>jspm</title>
<div class="map-container"></div>
<script src="jspm_packages/system.js"></script>
<script src="config.js"></script>
<script>
  System.import('app/main');
</script>
```

5. Import a module

Create a folder called 'app'; this will house our main JavaScript files. In a new file called main.js write the code below. We're making use of ES6's module loader as we're effectively saying 'I know that there's a file called Map.js at this location, import the Map export that it exposes'. Then we instantiate a new instance of it.

```
import { Map } from './Map';
new Map();
```

6. Install Google Maps

At the time of writing Google Maps does not provide a module-compliant API but sakren has written a wrapper for this so we can still use it within our app. Jspm's install mechanism is fairly flexible, it can look at its own internal registry, GitHub and npm. Note that we're providing a shorthand name of 'google-maps' to this.

```
$ jspm install google-maps=github:sakren/
node-google-maps
```

7. Import modules

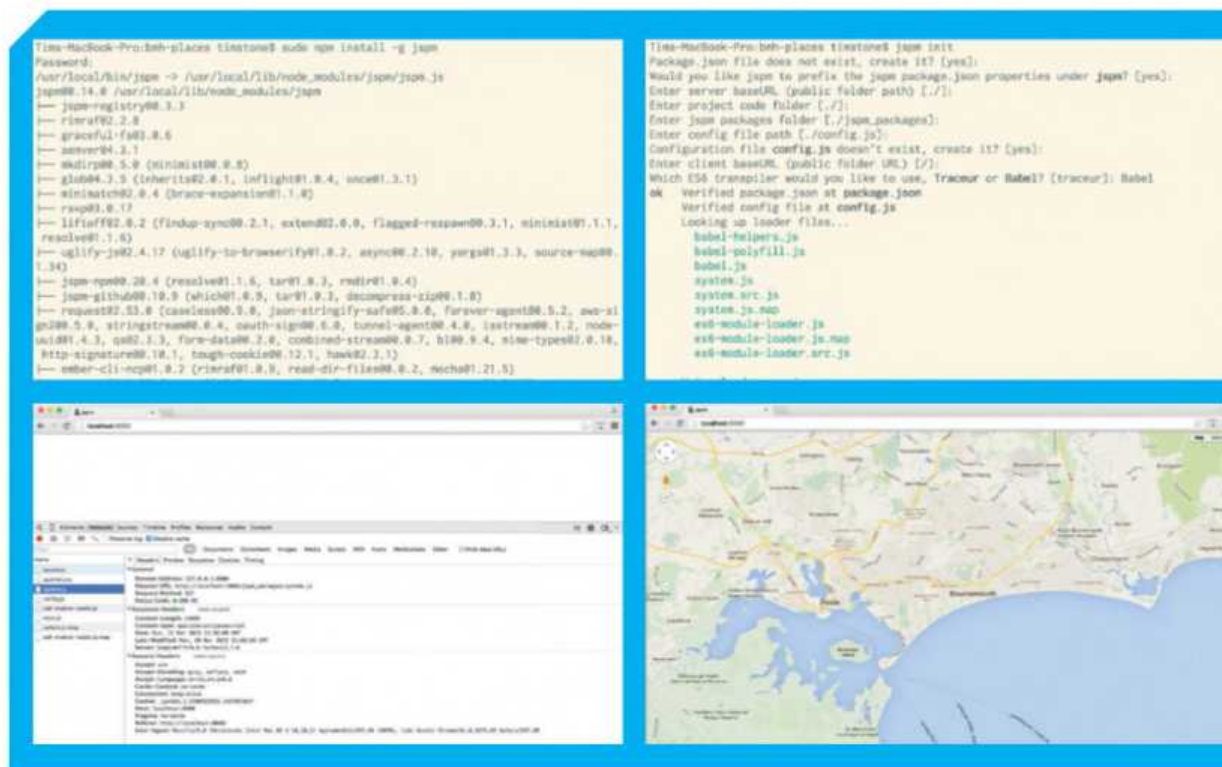
We'll import the GoogleMaps module we just installed and we'll also import a file called keys which we'll write ourselves. Note that when we reference a file on the system we prepend it with '.', this tells the loader where to look relative to this file. Once imported, default modules can be named anything you wish.

```
import GoogleMaps from 'google-maps';
import keys from './keys';
```

8. Store keys

Create a file called 'keys.js'. We're going to use a different

With its charting heritage, Highmaps content is also treated as a series of datapoints and the options for these are configured here

**<Top left>**

• Installing jspm is a cinch through npm. We're installing it globally as you'll probably want to use it across projects

<Top right>

• Initial setup of jspm is painless thanks to init as it configures your project to use package management

<Bottom left>

• The only manual loading that actually needs to be done within the HTML here is configuring SystemJS

<Bottom right>

• With the Google Map module installed and a little configuration we've got a map of Bournemouth rendering!

method of exporting this module just to show how flexible jspm is. This is the CommonJS way (which is probably familiar to you if you've used NodeJS before). We'll put any keys that we need here, and in this demo it's only the Google Maps one.

```
module.exports = {
  googleMaps: 'YOUR_API_KEY'
};
```

9. Map class

Our app has had an error because we've said that this file will export something called Map but so far we haven't. We'll rectify this by adding the following snippet to the bottom of Map.js. This creates an ES6 class called Map.

```
export class Map {
}
```

10. Constructor function

ES6 classes have a function called constructor that is run when the class is invoked. Within a class we don't have to use the function keyword as it's denoted by the parentheses. We're setting values for the GoogleMaps library and then loading; the function passed to 'load' will be called once it is loaded.

```
constructor () {
  GoogleMaps.LIBRARIES = ['places'];
  GoogleMaps.KEY = keys.googleMaps;
  GoogleMaps.load(this.createMap.bind(this));
}
```

11. Create the map

When Google Maps loads we call this function which will instantiate a new instance within the element provided and configured to be centred on Bournemouth. We invoked this with .bind(this) because the value of 'this' changes within the callback to 'undefined'.

```
createMap () {
  this.map = new google.maps.Map(
    document.querySelector('.map-container'),
    {
      center: {
        lat: 50.720806,
        lng: -1.904755
      },
      zoom: 12
    }
  );
}
```

12. Load buildings

As well as the load callback there is also the .onLoad event, which you can pass a function to, that is fired after the initial load callback. In a similar way to before, we're going to call a method called getBuildings with the current value of 'this' which points to the Map class itself.

```
GoogleMaps.onLoad(this.getBuildings.
  bind(this));
```

13. Import GetData

To help separate our application a little we're going to import a module to make the request for us. The

Debug with source maps

Babel includes source maps for the generated files so you can debug from within the browser without having to interpret any of the transpiled output.

GetData module we'll write will have a 'default' export as opposed to the named exports that we've seen previously (with the curly braces). Modules can export both default and named modules.

```
import GetData from './getData';
```

14. Install Papa Parse

The data that we want is in CSV format so we'll use a handy library called Papa Parse which specialises in all there is to know about parsing CSV. We install it with jspm and it's hosted on GitHub so we specify the username and repository name of the project.

```
$ jspm install github:mholt/papaparse
```

15. Import Papa Parse

Without having to include anything through HTML we're able to import the newly installed Papa module. We then create an object called GetData with a csv method, which will return the Papa.parse function and execute the passed callback. Papa Parse splits the data out for us and also deals with making the AJAX request. Note the 'default' keyword.

<Top left>

- Now we target all the p tags within our columns and give them default styles as well as 15 pixel padding to make a gutter

<Top right>

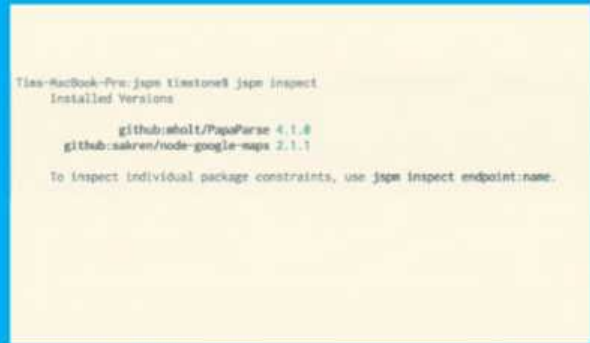
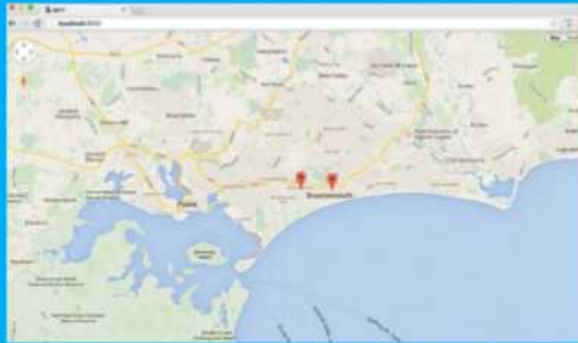
- With the column widths now added, we can see the grid taking shape with the columns positioned nicely

<Bottom left>

- We have now added in our outlines to show the grid in a nice visual way

<Bottom right>

- In this step, we will target all the paragraph tags again and align all the text and remove any margins or padding



Learn ES6

Babel has a very good guide for learning about ES6 features at babeljs.io/docs/learn-es6 but having a practical application is a good way to learn.

```
import Papa from 'PapaParse';
var getData = {
  csv: function (url, callback) {
    Papa.parse(url, {
      download: true,
      complete: callback
    });
  }
};
export default getData;
```

16. Get the buildings

We then use it as follows, we pass the location of the file we wish to download and the function to execute on completion.

Again we have to bind the current value of 'this' to the function and when we map out the data within 'process' the second argument effectively does the same thing as .bind(this).

```
getBuildings () {
  function process (results) {
    results.data.map(this.plotLocations, this);
  }
  GetData.csv(
```

```
    '/data/buildings_listed_buildings.csv',
    process.bind(this)
  );
}
```

17. Geocode results

The data in the file gives us a name which looks like 'Albert Road, Former Theatre Royal', but to place a marker on the map we need to know its latitude and longitude values. To do this we need to use the Google geocode service which comes with Google Maps.

```
plotLocations (result, index) {
  var geocoder = new google.maps.Geocoder();
  geocoder.geocode(
    {
      'address': result[1]
    },
    );
}
```

18. Geocoder response

If we get a result back from the geocoder then we immediately create a new marker (we'll write this method shortly). If it doesn't find a result then we'll try to find one using the Google Places API which is more advanced than the geocoder. Google limits how many places results we can make in a second so we stagger the requests with 'setTimeout'.

```
function (place, status) {
  if (status === google.maps.GeocoderStatus.
```

```
OK) {
  this.createMarker(result[1], place[0].
  geometry.location);
} else {
  setTimeout(
    this.searchPlaces.bind(this), 100 * index,
```



Does jspm add value to your project? Part one

There is no doubt that this approach requires more code to be downloaded and executed than if you were to simply write this in ES5. The fact is that it'll be years before these features are natively supported in enough client devices to make this viable in production. Even if you exclude IE there are still many mobile browsers that may never support it. You have to make a judgement on whether the flexibility this approach gives you works with your product or client. Once minified and concatenated, our project goes from 89 requests at 2.5MB to 74 requests at 1.2MB (this includes everything that Google Maps loads and .map files which inflates the number of requests a lot). This is a good starting point for long-term projects and will help introduce you to concepts that'll help you in the future.


```
result
});
}
}.bind(this)
```

19. Places configuration

SearchPlaces tries to find a location matching the passed query (eg 'Alum Chine Road, Westbourne Library') within a five-mile radius of the latitude and longitude passed to it (in this case, Bournemouth). By specifying this we ensure we don't get results for other countries! This is the initial configuration and the next step is the callback function.

```
searchPlaces (result) {
  var places = new google.maps.places.
  PlacesService(this.map);
  places.textSearch({
    query: result[1],
    location: {
      lat: 50.720806,
      lng: -1.904755
    },
    radius: 5
  }, /* next step */
}
```

20. Google Places response

If Google Places finds somewhere that matches then we just take the first result (there could be options) and we create a marker from that. Otherwise we output a message to the console. This uses ES6's string interpolation feature – a much cleaner way to embed variables in strings.

```
function (place, status) {
  if (status === 'OK') {
    this.createMarker(result[1], place[0].
    geometry.location);
  } else {
    console.log('Could not find ${result[1]}');
  }
}.bind(this));
```

21. Create markers

Now write the createMarker method. This creates a new Google marker with a title property so when the user hovers over a pin they can see what is there, and we also animate the pins to drop onto the map.

```
createMarker (title, position) {
  return new google.maps.Marker({
    map: this.map,
    title: title,
    animation: google.maps.Animation.DROP,
```



Does jspm add value to your project? Part two

Unlike HTML and CSS, JavaScript is not backwards compatible by design. This means that it is up to us as site builders to decide where we draw the line in the sand with regard to support. Many of us have already decided that IE7 and below do not warrant our time and those users have a very broken experience using the web. At some point we'll have to make the decision when to stop using a transpiler like Babel, which will affect the browsers we support today. For now though, one of its greatest strengths is that there is no in-between compilation step, it all happens transparently until you need to bundle your application. Although, as protocols and browsers become smarter, new ways of supporting these older browsers may become transparent.

```
position: position
});
}
```

22. Bundle for production

Our app is finally code complete! Unfortunately with all of those modules being loaded it's making far too many requests. Jspm has a 'self-executing' bundle process which combines and minifies all of the ES6-to-ES5 features and all of the modules into a single streamlined bundle.

```
$ jspm bundle-sfx app/main --minify
```

23. Not just JavaScript

The frontend isn't all about JavaScript, though. Jspm enables you to also install CSS files like Normalize or Bootstrap too. This will then include the CSS-loading plugin which does add some overhead. Something to note is that CSS ordering is not guaranteed so modular styles must be 'name spaced', either by a base class or unique ID.

```
$ jspm install normalize.css
```

24. Import normalize.css

To include a CSS module you use it identically to a JavaScript one. This makes including only the CSS that is used on a per-page basis much easier. Coupled with HTTP/2's features (where bundling will become an antiquated antipattern) this should lead to performance improvements.

Package.json for jspm

The configuration options for managing all of the modules in your project are extensive and a great explanation of how they work can be found on GitHub at bit.ly/1DJOKyk.

```
import 'normalize.css';
```

25. Inspect and update

To see what packages are installed, and their dependencies, we can use 'inspect'. This will print the name and version of all the installed packages. As you'd expect from a tool like this you can also update all packages at once. Jspm will look at the package.json file to ensure only supported versions are installed, however multiple versions of the same package are supported.

```
$ jspm inspect
$ jspm update
```

26. Module maintenance

Jspm gives you very granular control over what versions of packages are installed and it's not rare to find situations where some libraries use older versions. You can manually override this by updating config.js. This can sometimes leave orphaned packages, those that are installed but no longer used and they can be purged with 'clean'.

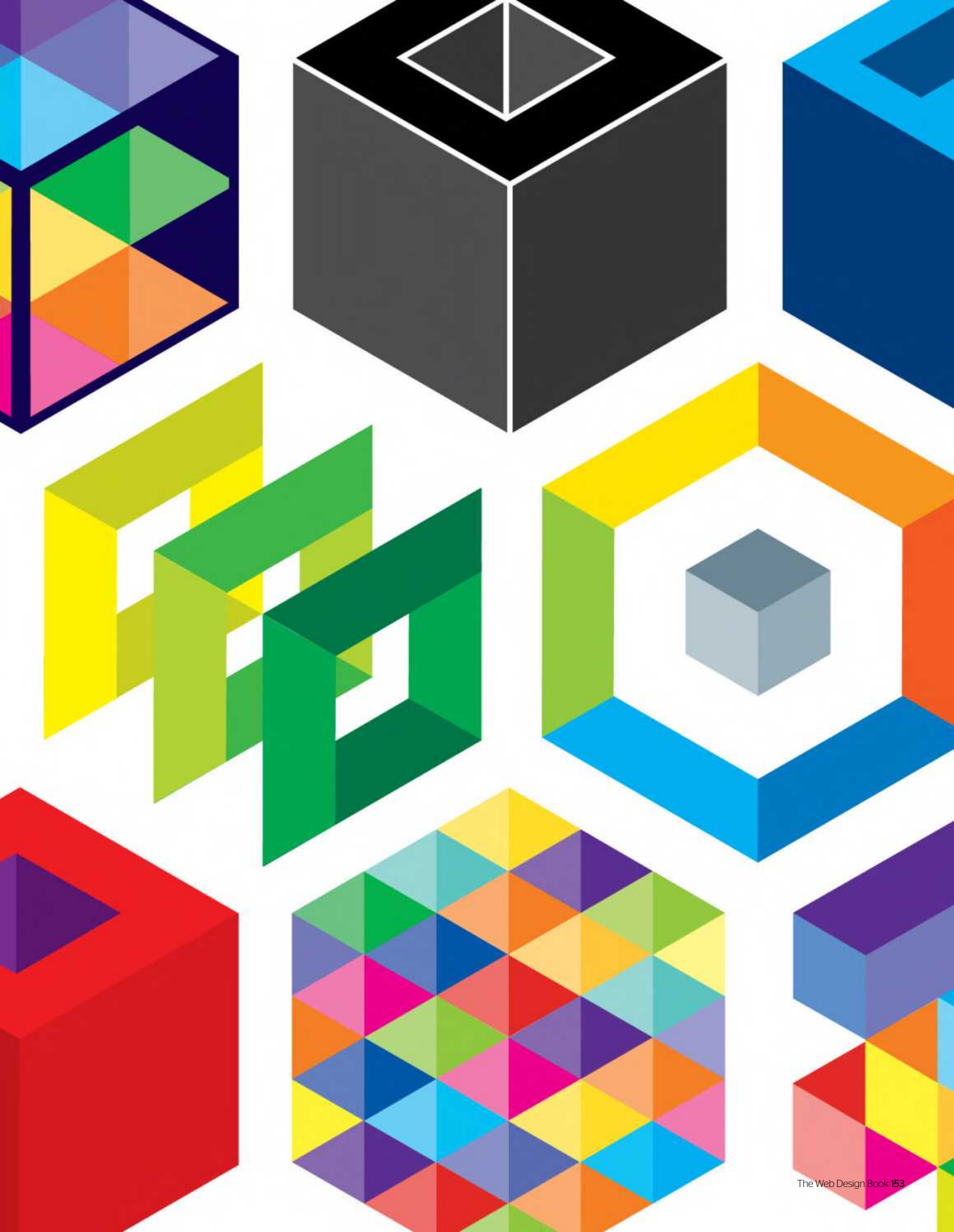
```
$ jspm clean
```

“ This can sometimes leave orphaned packages, those that are installed but no longer used ”

Mobile Apps

- 154** Build apps with Facebook's React native framework
Create a cataloging image app
- 158** Create a reactive web app using Meteor
Share your creations
- 164** Code real-time presentation applications with Socket.IO
Master bidirection communication
- 168** Design and share a web app faster with React
Harness your server power
- 172** Make a Chrome OS app with JavaScript
Access nonstandard APIs
- 176** Build a mobile web app with NativeScript's library
Make apps for iOS and Android
- 180** Code a PhoneGap memo app with photos
Pass files to your phone quickly
- 184** Develop a reactive web app with Angular-Meteor
Combine Meteor and Angular

☞ So, you've been writing JavaScript for a little while now. Thing is, no matter how awesome the web is (and it is fully awesome) the call of the native platform still beckons ☞



↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Build apps with Facebook's React Native framework

Create a JavaScript-powered native cataloguing image app with the React library





o, you've been writing JavaScript for a little while now, you'd say you've gotten quite good at it, you've made websites, a couple of small libraries and even a few web apps. Thing is, no matter how awesome the web is (and it

is fully awesome) the call of the native platform still beckons. You decide to give it a crack, but you've never written anything with Objective-C before – sure, there's this new Swift thing that looks familiar, but the process of building an app is still foreign to you. We could try some transpilers, that is, write some apps in JavaScript and then have something like Titanium figure out what your code is trying to do, but that's a painful process to get right. We could try out PhoneGap – that's nice and familiar and you can get great results after some tweaking, but JavaScript optimisation is hard – especially when it's inside a native wrapper. What are we to do?

Wouldn't it be great if we could write apps in JavaScript with our favourite IDE but have everything in our views be native – wouldn't that be ideal? Well, that's React Native – a supercool framework from Facebook that lets you write your app logic in familiar React.js syntax and then renders all of the view using native iOS/Android elements. It's not transpiling or reinterpreting your JavaScript code, it's running it separately in the background. In this tutorial, we're going to be making a catalogue app, that is, an app that shows us cats (see what we did there?), so let's get to it!

1. Get React installed

React Native is a complicated piece of kit, so there are quite a few dependencies but thanks to homebrew, NPM and Node.js setting up React Native is simple. To install just run these commands in a terminal window:

```
brew install watchman
npm install -g react-native-cli
```

2. Create a new project

Now we have the React Native CLI tools installed, CD to a directory that you want to create your React Native project in and run 'react-native init catalogue'. This will create a bare-bones project with everything we need.

3. Fix Xcode project/iOS simulator

For some, the Xcode project might not work out of the box, Xcode can fix that for us. Open the Catalogue.xcodeproj file inside the folder you just created. Xcode will open. Click on the project tab on the left and select 8.0 from the Deployment Target drop-down menu.

4. Select device

Look at the play button in the top left of Xcode, just to the right of it there should be a small icon with iPhone 4s or something similar, click it and select iPhone 5s or iPhone 6 (depending on your version of Xcode) from the dropdown that appears and hit play.

5. Fix errors

Click on the tabs showing the error and select the first fix that's suggested. Hit play. The iOS simulator might throw an error so select 'iOS simulator' in the menubar and click 'reset content and settings' and hit play again.

6. Our bare-bones app

In our iOS simulator, our app will now be open. Go back to the catalogue folder we created and open index.ios.js in your favourite IDE. This is the main entry point for React Native to run our app from. We're using the virtual DOM to create elements, but instead of DOM elements, we're using equivalent iOS elements.

7. Clear the house

This React Native project has one view, but we need two views for this tutorial: one for listing our categories, and another for viewing the photos in that category. Delete everything in index.ios.js and insert the following in its place to import the required modules for our project.

```
'use strict';
var React = require('react-native');
var {
  AppRegistry,
  StyleSheet,
  Text,
  Image,
  ListView,
  TouchableHighlight,
  NavigatorIOS,
  View,
} = React;
var CatViewing = require('./catViewing');
var List = require('./list')
```

8. Create a starting point

We now have everything we need except for a view. Add the following code to index.ios.js, it uses the NavigatorIOS module to let us create views, this means we get a lot of things like back buttons and animations between views for free. The initialRoute property determines where our app should find the code for our first view.

```
var catalogue = React.createClass({
  render: function() {
    return (
      <NavigatorIOS
        style={styles.container}
        initialRoute={{
          title: 'Cat-Alogue',
          component: List,
        }}
      />
    );
  }
});
```

9. Stylings

Let's create a simple stylesheet. React doesn't use CSS, it

uses a JavaScript polyfill to emulate a limited subset CSS with native components, and as such we define styles like so. AppRegistry exposes our React class to our native code, it's our true entry point to the application.

```
var styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row',
    justifyContent: 'center',
    // alignItems: 'center',
    backgroundColor: 'white',
  },
});
AppRegistry.registerComponent('catalogue',
  () => catalogue);
```

10. Create list.js

In index.ios.js we defined the List component as our first view. Create the file 'list.js' in our project folder and a file called 'catViewing.js'. Import the following modules:

```
'use strict';
var React = require('react-native');
var {
  AppRegistry,
  StyleSheet,
  Text,
  Image,
  ListView,
  TouchableHighlight,
  View,
} = React;
var cats = require('./cats'),
thumbCats = [];
var CatViewing = require('./catViewing');
```

just after that, we're going to create the List class and all of the functions that it will use to build our view:

```
var list = React.createClass({
  getInitialState: function() {
  },
  renderCatThumb: function(cat) {
  },
  loadCategory: function(category) {
  },
  render: function() {
```

Learn once, write everywhere

We've all heard of 'write once, deploy everywhere' but there's rarely an instance where that *actually* works out. Facebook's suggested philosophy is "learn once, write everywhere". Once you've learned React, you can write it for any platform, rather than maintaining one codebase that has to accommodate everything.

If we run our code, we get an error, this is because it can't find the `cats` module we defined earlier. You can grab `cats.js` from [FileSilo](#) and add it to our project folder, it's a really simple JavaScript module that contains URLs for the cat images we'll be using.

We need to help React make sense of our data for iOS so we use a `ListView` to list categories and a `ListView` will only accept a `DataSource` object as an argument. Inside of `getInitialState`, add the following code to create an object for each of our categories for making a tab:

Is it honestly native?

Yes! Facebook have yanked out the JavaScript engine from the WebViews and run it in a separate thread. Your app logic runs in JS but everything you see on the screen is completely native.



- Facebook doesn't want your code to fail silently. It wants to get up in your face so you write better code, so here is what they've dubbed the 'Red Screen of Death'

- Facebook doesn't want your code to fail silently. It wants to get up in your face so you write better code, so here is what they've dubbed the 'Red Screen of Death'

1. **Introduction**




```
...
var catsToView = cats[category];
this.props.navigator.push({
  title: category + " Cats",
  component: CatViewing,
  passProps: {catsToView},
});
...
```

18. catViewing.js

With `loadCategory()` we've passed everything we need to render our images, but where did we pass it to? What about the `CatViewing` class we're about to create in `catViewing.js`? `CatViewing` works just the same as our list class worked, except instead of being a `ListView`, we're using a `ScrollView` which gives us a little more flexibility when it comes to how we lay content out. Add the following code to `catViewing.js` to get started:

```
'use strict';
var React = require('react-native');
var {
  Image,
  PixelRatio,
  ScrollView,
  StyleSheet,
  Text,
  View,
} = React;
var CatViewing = React.createClass({
  render: function() {
    },
  });
module.exports = CatViewing;
```

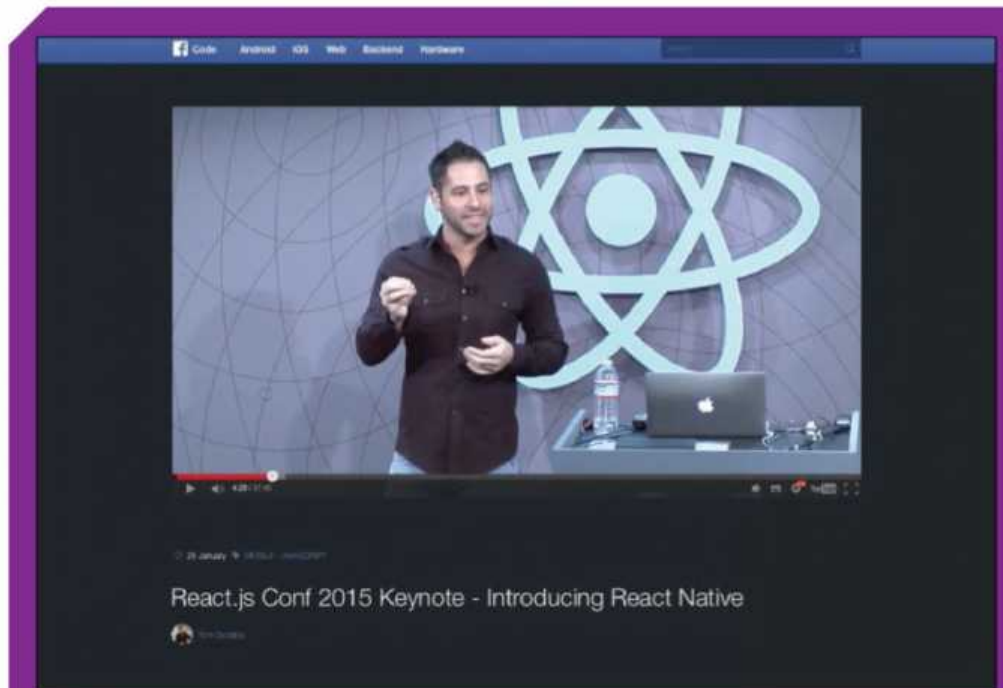
19. Get our list of cats

The first thing we want to do is get a reference for our list of cats that we passed to the view from our `List` class. Each `React` class has a `props` object which we can access by using `this`. We don't need many functions here for our `CatViewing` class, in fact we only really need one and that is a `render()`. So amend `render()` to look like the following:

```
render: function() {
  var catsToShow = this.props.catsToView;
  return (
    <ScrollView contentContainerStyle={styles.
      contentContainer}>
      {catsToShow.map(createThumbRow)}
    </ScrollView>
  );
},
```

20. Map our cats

In our `scrollView`, we have mapped `catsToShow` to `createThumbRow`, but we haven't written that yet! So the actual making of the `createThumbRow` is a very simple process - for each image we pass to it, it will create an instance of the `Picture` class we're about to create. `Picture`



JsCSS - WAT?

Aligning objects on a screen is a tough business, just ask any iOS developer who's done a lot of manual positioning, it can be a nightmare. Facebook looked around for a solution to aligning things intuitively and it found one, CSS! Problem is, implementing CSS for iOS was not a prospect they fancied much, instead, they opted to create a JavaScript subset of CSS styles that could be used to position native iOS elements in and around native iOS views so it has all the goodness of CSS and JavaScript and full speed of a native. One of the coolest things about JsCSS? It has `Flexbox`, not the full `Flexbox`, but something very close and easy to use. Check out the Facebook Developers YouTube channel for more information on `React Native` bit.ly/1AK9zmZ.

is a very simple class, so it's very similar to `CatViewing` in that sense, it only has a `render` function and all it does is take a wrap for an image in a view and then it will grab that image's source. Just think of it as if you are wrapping an `` in a `<div>` and then setting the `src` attribute over in HTML.

```
var createThumbRow = (uri, i) => <Picture
  key={i} uri={uri} />;
var Picture = React.createClass({
  render: function() {
    return (
      <View style={styles.centering}>
        <Image style={styles.imageHolder}
          source={{uri:this.props.uri}} />
      </View>
    );
  }
});
```

21. Finish up

We're almost ready to view all of our cats. If you refresh the iOS simulator, we get a nice big error (we might even crash Xcode!) because we haven't added styles for the `Picture` or `CatViewing` classes. But, that's simple, if we just add the following after our `Picture` class closes, we're all done. Now we can

take our `Cat-alogue`, and put it wherever we want it to be.

```
var styles = StyleSheet.create({
  contentContainer: {
    padding: 10,
  },
  centering : {
    alignItems : "center"
  },
  imageHolder : {
    width : 280,
    height : 280,
    marginBottom : 5
  }
});
```

22. Fade out

In the `'draw'` function, comment out the background colour. Set the fill colour to black with a low opacity of around ten per cent. A rectangle is now drawn over previous frames with the low opacity, causing them to fade out. Save and test to see this final effect.

```
background(0);
fill(0, 25);
rect(0, 0, windowWidth, windowHeight);
```

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Create a reactive web app using Meteor

Share your creations and prototypes in very few lines of code with the powerful open-source platform **tools | tech | trends** Meteor, JavaScript, HTML



eteor has matured a lot in recent months, having finally, and rather crucially, reached the milestone of version 1 (we'll be using 1.0.2.1

to complete this tutorial). Every aspect of Meteor has markedly improved, from the homepage

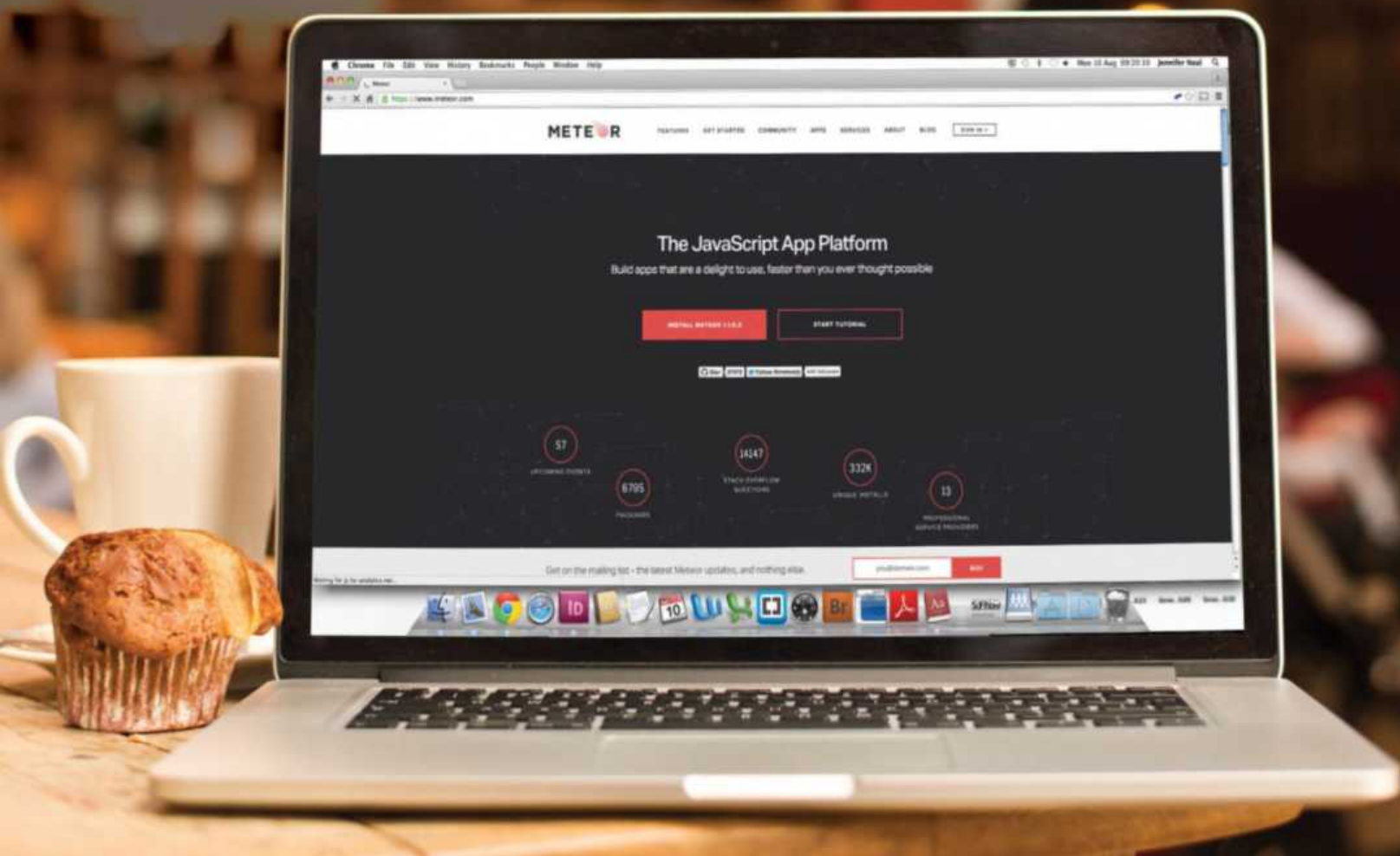
and documentation to the community and codebase itself.

Meteor is not a single library, Meteor is a host of projects, which include: the software libraries, tools such as the build tool, Isobuild, standards formalisation, and services like the official package server.

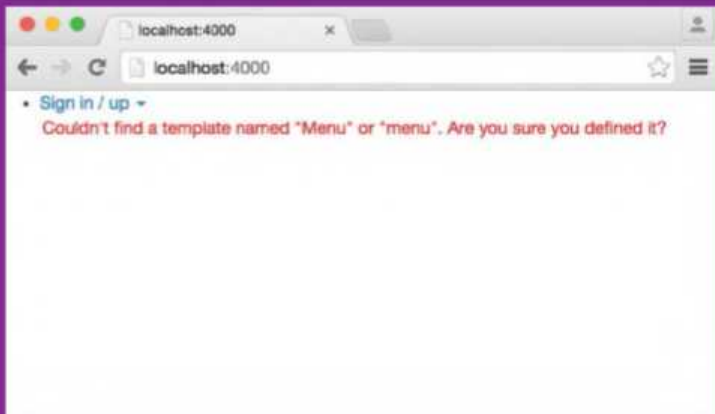
Meteor projects run on Node and use MongoDB to store data. As it's all written in JavaScript, the same

code can run on the client and server. Meteor uses this to create 'reactivity', namely that changes are reflected immediately and propagated to every connected user.

To highlight the changes to Meteor, we'll build a basic ordering system for a fictional restaurant. This exercise will introduce you to Iron Router, adding packages, database communication and reactivity to name a few.



Mobile Apps



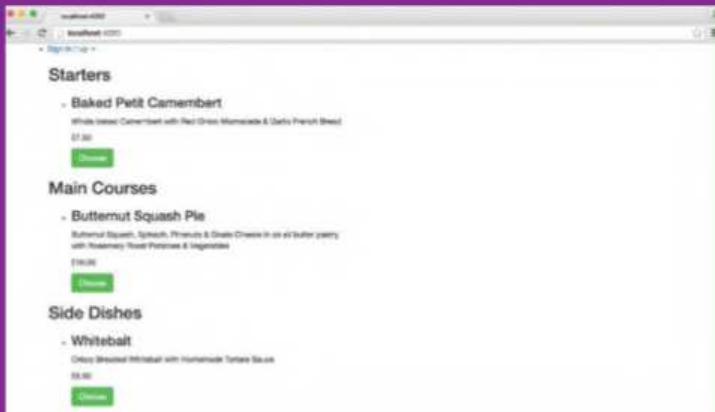
<Above>

- Our app is rendering correctly and even provides helpful hints in the UI when things aren't working like they should be



<Above>

- Our menu is now displayed! As soon as you make changes to the database it'll be pushed through to the client



<Above>

- The formatted price is now working as intended and displaying the other menu items with the menu item helper



<Above>

- Our finished kitchen interface updates as soon as customers place an order, which the chef can then check off

Atmosphere and packages

Packages extend Meteor with great functionality: at the time of 0.8 Atmosphere (<http://atmospherejs.com>) wasn't officially part of Meteor, but now support is baked right in.

6. Add packages

Packages are first or third-party libraries that add functionality to your project. We're going to add Iron Router (a routing library for Meteor), Twitter Bootstrap for styles, an accounts manager, a spinner

```
<main role="main" class="row-fluid col-md-6">
  {{> yield}}
</main>
</div>
</template>
```

to show while loading, and the Underscore utility library.

7. Configure routes

Within the lib folder create a file called 'Router.js'. The configuration block tells Iron Router to use the 'layout' template as the default layout for each page.

Add routes by calling the .route() method and passing the URL string to match as well as the name of the associated layout file. We're also overriding the default layout template on the kitchen route so it won't inherit that layout.

```
Router.configure({
  layoutTemplate: 'layout'
});

Router.route('/', {name: 'menu'});
Router.route('/kitchen', { name: 'admin',
  layoutTemplate: null
});
```

8. Create a header template

The header template doesn't exist yet, so let's create an empty template folder in the 'includes' folder.

'loginButtons' is added by the accounts-ui package – it comes with Meteor and adds a log-in system out of the box!

9. Create a menu template

We've got various parts rendering now, but they can't find the menu! Let's rectify that. Create a file called 'menu.html'. To repeat the block, simply replace Starters with the property name like main_course, for example.

10. Place the menu

Where does the menu come from? We're going to store the menu in the MongoDB as a collection. Within lib/collections, create a file called 'menu.js'. Then create another file called 'publications.js' in the server folder. This will tell the server to enable the client access to the menu collection and give both the client and server access to the Menu variable.

```
// lib/collections/menu.js
Menu = new Mongo.Collection('menu');
// server/publications.js
Meteor.publish('menu', function() {
  return Menu.find();
});
```

11. Get a menu helper

Create another file called 'menu.js' within templates/menu. The second part of the property (Template.foo) is the name of the template. Helpers are variables within the template, so use this helper like {{menu}}. It's returning a reference to the Menu collection and it returns all of the menus that it can find.

```
Template.menu.helpers({
  menu: function () {
    return Menu.find();
  }
});
```

12. Insert menu data

Next, add the menu to the database. Within a browser console follow the GitHub Gist link and copy and paste the JSON between the parenthesis. As soon as you do this, the view should update with the new menu data from the database.

```
Menu.insert(
  // JSON from http://bit.ly/1vE0Fo0
);
```

13. Create the menu item template

Unfortunately you still won't see anything! Why? We referenced a template called menuItem that we haven't created yet. But once this is done, the menu will be displayed beautifully. Within templates/menu create a file called 'menu_item.html'. As you would with other template systems, reference object properties with the double curly brackets.

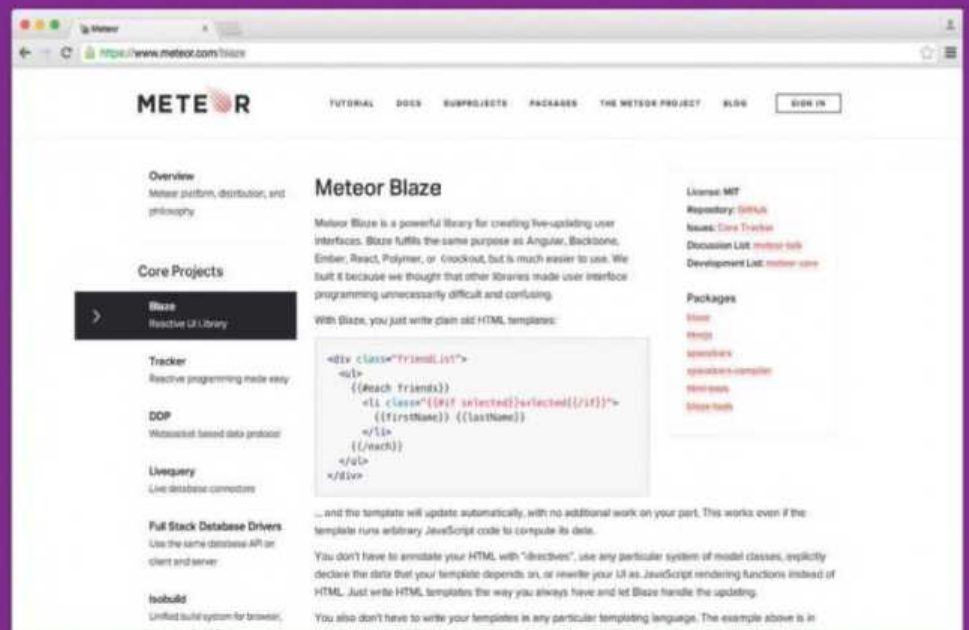
14. Fix the prices

Hooray, our menu is rendering! But no price is showing up, that's because instead of just outputting {{price}} we're outputting {{formattedPrice}}. As it comes through as a number we should format it as a proper, consistent price. Create a file called 'menu_item.js'. 'this' references the actual object from the menu collection and toFixed transforms the number from 6 to 6.00.

```
Template.menuItem.helpers({
  formattedPrice: function () {
    return this.price.toFixed(2);
  }
});
```

15. Template events

Events to templates are added in a novel way. Each listener is the event followed by a CSS selector. This



Reactive rendering with Blaze and Tracker

A part of Meteor that's integral to how it works, but not yet mentioned, is the UI library Blaze. The library includes Spacebars and a number of other packages for dealing with HTML and JavaScript, so you don't have to worry about dealing directly with the DOM in most cases, you just write your template. Blaze isn't tied down to just Spacebars, though. If you prefer writing your markup in Jade then it supports that too, it's simply a way of creating a reactive interface. It does this by utilising a library called Tracker which enables 'transparent reactive programming'. Tracker was originally called DDP, which we touched on briefly last time in issue 224. Tracker enables any library to register what dependencies to listen for and be notified when they change.

Our button is going to change behaviour depending on if it has a class called choose or not

makes them highly readable and ensures that they're all kept in the same place. The event is passed through and we can reference the current item as 'this'.

```
Template.menuItem.events({
  'click button': function (e) {
    /* next step */
  }
});
```

16. Add an order

Our button is going to change behaviour depending on if it has a class called choose or not. If it does then we're going to create a new order by inserting into the Orders table. We're getting the table information from the logged-in user (each table could have its own account) so the kitchen knows who ordered what.

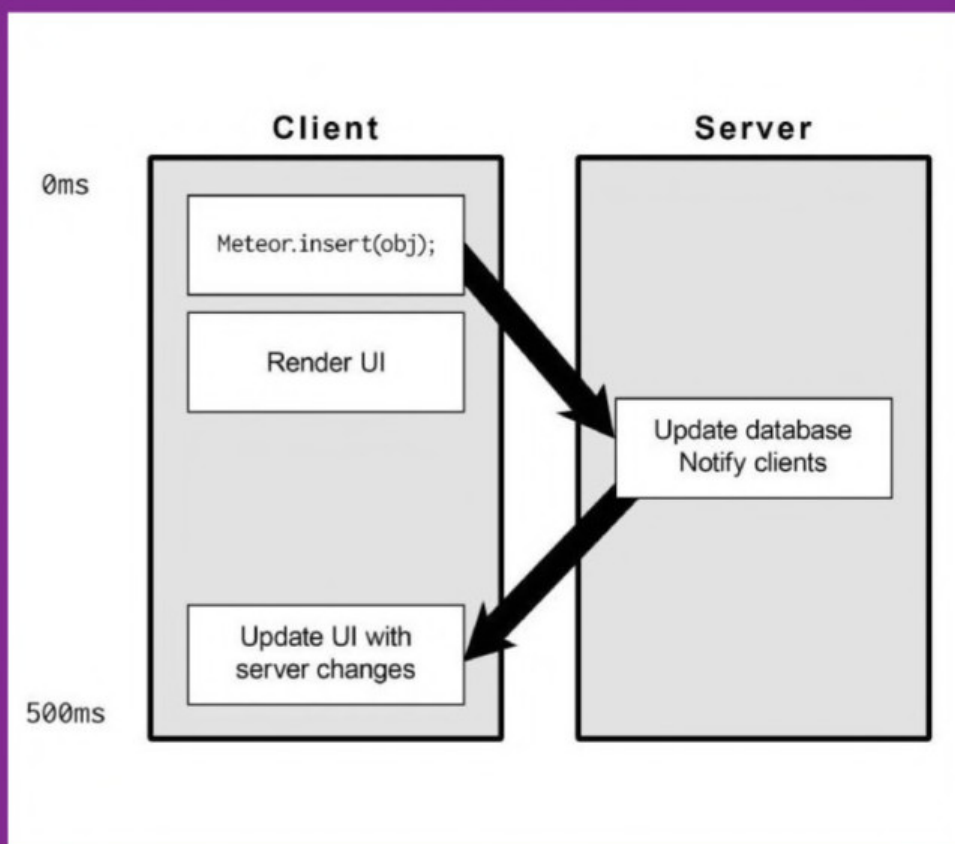
```
if (e.target.classList.contains('choose')) {
  e.target.innerText = 'Remove';
  this.orderId = Orders.insert({
```

```
table: {
  name: Meteor.user().username,
  id: Meteor.userId()
},
item: this
});
} else {
  /* next step */
}
e.target.classList.toggle('choose');
```

17. Remove an order

When the user clicks the button we toggle the class of 'choose' and show Remove text instead. Removing an item from the database is just as easy as adding one, call remove with the ID of the record you wish to remove, optionally you can have a callback to handle errors and successes.

```
var self = this;
Orders.remove(this.orderId, function (error)
{
```



Built-in latency compensation with Meteor

This sounds like a complicated process but Meteor makes it seamless. When you make a change to the database, that change is performed on the client immediately and then when the database comes back with a response, it then updates the client and UI. This behaviour is part of the reactivity built-in to Meteor and makes Meteor apps perceived performance faster than it actually is because the user always gets an immediate response. In the case of collections you can disable this on a case-by-case basis with `{ reactive: false }` when you set up `find()`, eg with `Collection.find({}, { reactive: false })`. This is also useful if you know the source won't change.

Deploying your Meteor project

Meteor apps can now be deployed using services other than the official 'meteor deploy', like Nitrous, Modulus, and Meteor Up.

```
if (error) {
  alert(error.reason);
} else {
  alert(self.name + ' was removed.');
```

18. Secure the app

Meteor includes a couple of convenience packages, which make it easy to get it up and running quickly, but it shouldn't be used in real projects - these are `autopublish` and `insecure`. `Autopublish` publishes the entire database to all clients, and `insecure`

enables all database writes by default. Let's remove those.

```
$ meteor remove autopublish
$ meteor remove insecure
```

19. Subscribe to menu

Uh oh, our menu has disappeared! This is because although the server is publishing the menu we're not subscribing to it client-side. Create yet another file called `main.js` in `client/templates`. This subscribes to not just the published menu, but a collection that we haven't created yet called `orders`. As soon as you save this, the menu will reappear.

```
Meteor.subscribe('menu');
Meteor.subscribe('orders');
```

20. Kitchen template

Our menu now successfully places orders, but our fictional kitchen needs to see the orders as they come through. First we'll create a new template to show these

☞ We'll output the item's name and a checkbox that could be used to notify the patron when their food is cooked ☞

orders called `'kitchen.html'`. `{{table}}` is the name of the table (the username, we've called my users `'Table 1'`, `'Table 2'` and so on). An exclamation mark denotes a Spacebar's comment.

```
<template name="kitchen">
<div class="container kitchen-view">
<h1>Orders</h1>
<ul>
  {{#each tableOrders}}
<h2>{{table}}</h2>
  {{! next step}}
  {{/each}}
</ul>
</div>
</template>
```

21. List orders

We'll simply output the item's name and a checkbox that could be used to notify the patron when their food is cooked or simply as an internal reference. We could also limit the fields returned from the server within the helper if keeping requests lean were a priority.

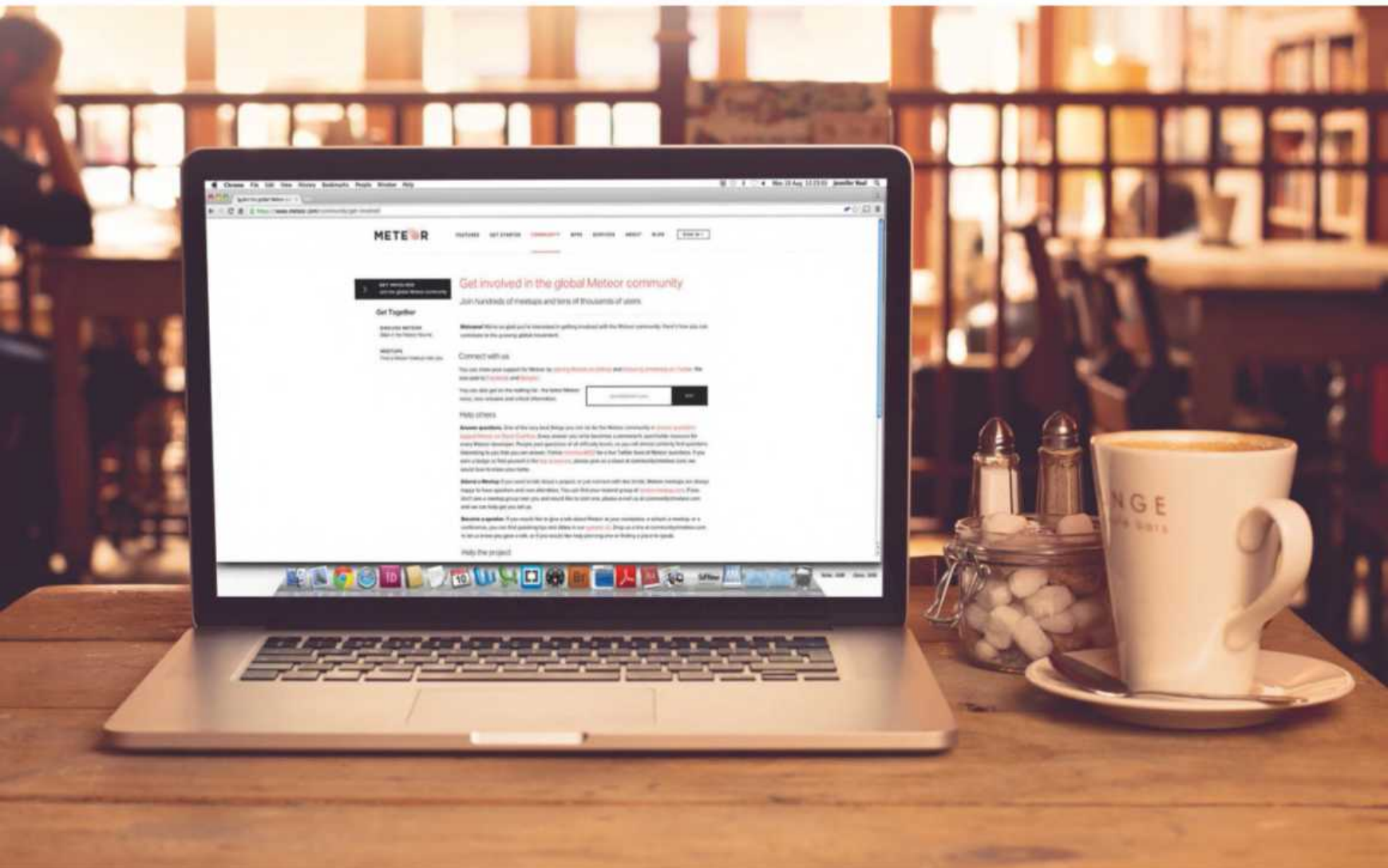
```
{{#each orders}}
<li>
<label>
<input type="checkbox">
  {{item.name}}
</label>
</li>
{{/each}}
```

22. Kitchen helper

We've seen `Collection.find()` a few times but this doesn't actually get the records. It returns a 'cursor' to it, `'fetch()'` gets the records and because a subset of the database is replicated on the client, it's instantaneous. We are then grouping each order by the table that it's associated with using Underscore's `groupBy()` method.

23. Map table data

We then use another Underscore method called `.map()`, which transforms each item in the array so we end up with a structure that looks like `'[{ table: 'Table 1', orders: []`



], ...]'. We can then iterate over this in our kitchen template and easily output each tables' orders. The first argument is the array of orders and the second is the key.

```
return _.map(groupedTables, function(orders,
table) {
  tableOrders: function () {
    return {
      table: table,
      orders: orders
    };
  };
});
```

24. Order permissions

In a new file under `lib/collections/orders.js` we're going to add another Mongo collection. We're also specifying that the client can insert or remove an order, but not update it. Instead of just returning true you could add specific instructions like `'return Meteor.user().roles.indexOf('admin') > -1'` (note that we'd also need to use the `meteor-roles` package).

```
Orders = new Mongo.Collection('orders');
Orders.allow({
  insert: function() {
    return true;
  },
  remove: function() {
    return true;
  }
});
```

25. Publish orders

Within `server/publications.js` we need to add another publish block beneath the menu one to publish the orders collection. Meteor includes a pared down version of Mongo on the client known as `MiniMongo`, which is why we can use Mongo commands client-side.

```
Meteor.publish('orders', function() {
  {
    return Orders.find();
  }
});
```

26. Loading indicator

When we added the Meteor packages for this project we added one called 'spin', which is a simple package that shows a CSS spinner. We'll use it to show the user we're doing something while it loads menu data. Create a file called 'loading.html' - it contains the template that points to the spinner partial.

```
<template name="loading">
  {{> spinner}}
</template>
```

27. Configure Iron Router

Finally we configure Iron Router to show the loading template when it's waiting to subscribe to the menu. You can also return an array like `return ['Meteor.subscribe('menu'), Meteor.subscribe('orders')]` if you wanted to wait for both.

We now have a system that takes orders and automatically updates the kitchen when they arrive. And you're done!

Build real-time presentation applications with Socket.IO

Master bidirectional communication with an interactive presentation app using Angular and Node.js

tools | tech | trends HTML, Node.js, AngularJS, Socket.IO



Since its inception Node.js has been designed to be event-driven. This means that Node is very efficient at listening and responding to events.

The frontend of an application is a different story. Until recently the only way to update a page was either to refresh the page or ask the data for new data. WebSockets and

server-sent events have blown this paradigm out of the water and enabled two-way transfer between client and server. We're going to look at a library that makes dealing with this type of bidirectional communication as painless as possible. Socket.IO is supported by a company called Automattic.

It's a hugely popular package and is the most depended on package NPM module, with good

reason. It's been around for five years and is still under active development.

To explore how to create a real-time web app, we're going to build an HTML-based presentation viewer so everyone's screen updates at the same time and anonymous questions can be posited to the presenter. We're going to be using AngularJS on the frontend but it isn't a requirement of Socket.IO.

1. Scaffold the server

We're going to start by using Express, a web framework which simplifies many things about Node including server creation and routing. Then we will use express-generator, which will create a skeleton app for us; we'll call ours 'preso'. Then install the dependencies and run it in debug mode.

```
$ npm install express-generator -g
$ express preso
$ cd preso && npm install
$ DEBUG=preso ./bin/www
```

2. App structure

You should now have a number of files and folders created for you. In the bin folder 'www' ties the application together with the previous command. Node_modules contains the server dependencies, public will contain static assets to deliver to the client, routes contains code for each route, and views comprises the template files.

```
!"" app.js
!"" bin
!"" node_modules
!"" package.json
!"" public
!"" routes
!"" views
```

3. Install Socket.IO

Next install Socket.IO via npm. This may take a while as it fetches, downloads and compiles everything it needs. Socket.

IO's core is Engine.IO, which you can use if you just require the transport layer. Socket.IO has additional features like reconnection logic and gracefully degrades on the client, whereas Engine.IO progressively enhances.

```
$ npm install socket.io save
```

4. Create events.js

Now create a new file called 'events.js' at the root level (same as app.js), which will hold our Socket.IO code. Let Node know what reading the file is going to return and do this with 'exports' so that this file (module) will export a function. Then we require the Socket.IO library.

```
module.exports = function (server) {
  var io = require('socket.io')(server);
};
```

5. Require events

Within the 'www' file require the events file we just made after the line 'server.on('listening', onListening);'. We then pass it the server instance that Express creates on line 22 ('var server = http.createServer(app);' as Socket.IO needs to be told where to attach itself to. So we're saying, load this file and as we know it exports a function, pass 'server' to it.

```
/** * Attach Socket.IO listeners. */ var
events = require('../events')(server);
```

6. Master layout file

By default Node uses a template engine called Jade, which it compiles to HTML; Jade is a succinct way of

describing HTML. In views/layout.jade we require some CSS files and Socket.IO automatically creates a client-side JavaScript file.

```
doctype html
html
head
  title= title
  link(rel='stylesheet', href='//maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min.css')
  link(rel='stylesheet', href='/styles/main.css')
block content
  script(src='/socket.io/socket.io.js')
```

7. Use index.jade

Layout.jade gives a reusable outer shell and any other content is rendered in 'block content'. Use this with the 'extends' keyword. This tells Jade to use layout.jade and insert the following HTML where it sees block content. Now make a body tag with some Angular attributes.

```
extends layout
block content
  body(data-ng-app="presentationApp", data-ng-controller="PresentationController")
```

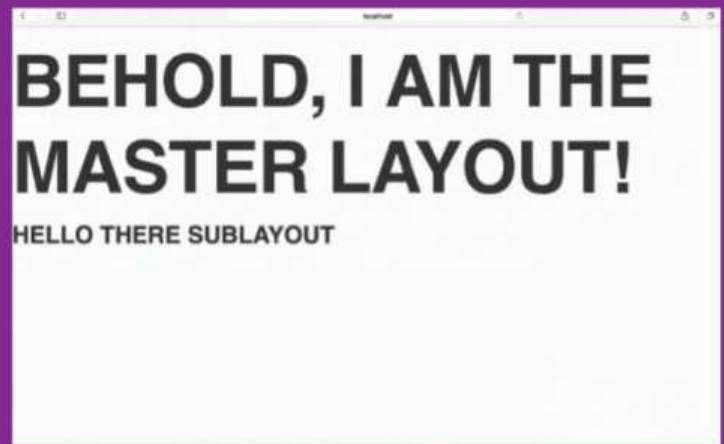
8. Install Angular

Our preference is to use Bower, and you can download a local copy or link to it on a CDN. We'll also use the Angular animate module to handle slide transitions. When Bower is installed, do the following:

```

2. bash
Tims-MacBook-Pro:test timstone$ express preso

create : preso
create : preso/package.json
create : preso/app.js
create : preso/public
create : preso/public/javascripts
create : preso/public/images
create : preso/routes
create : preso/routes/index.js
create : preso/routes/users.js
create : preso/public/stylesheets
create : preso/public/stylesheets/style.css
create : preso/views
create : preso/views/index.jade
create : preso/views/layout.jade
create : preso/views/error.jade
create : preso/bin
create : preso/bin/www
  
```



<Above>

•The sublayout template is also displaying, which means they are all working! This is where the directive will show the slides

<Left>

• Installing and scaffolding a new Express app takes seconds thanks to express-generator doing the work for us

```
$ cd public
$ bower install angular angular-animate
```

9. Work on the structure and module

We're going to split the frontend of our application into four folders: controllers, directives, modules and templates. There will only be one file in each but if you wanted to extend this project then you won't have to restructure it. Start by specifying all of the modules our app will depend on.

```
angular.module('presentationApp', [
  'ngAnimate',
  'presentationController',
  'slidesDirective'
]);
```

10. Presentation controller

Next we'll create the presentation controller. This will contain the logic to move the slides back and forth, but for now we'll simply set up a connection to our WebSockets server. We can do this by passing '/' to connect, this will default to the site address and it works no matter if you're connecting via localhost, IP address or URL.

```
angular.module('presentationController', [])
.controller('PresentationController',
[ '$scope', function ($scope) {
  var socket = io.connect('/');
}]);
```

11. Add 'slides' directive

Next we'll start writing the slides directive. After the body tag in index.jade add a div tag with three attributes: data-slides so that Angular sees that it should place the slides directive template here, data-path which is the name of the folder we'll attempt to read the slide files from, and data-amount which is the amount of slides we have.

```
div(data-slides, data-path='slides' data-amount='3')
script(src='/bower_components/angular/angular.js')
script(src='/bower_components/angular-animate/angular-animate.js')
script(src='/scripts/controllers/presentation-framework.js')
script(src='/scripts/directives/slides.js')
script(src='/scripts/modules/presentation-framework.js')
```

12. Slide directive functionality

Our slides directive will replace the div with the data-slides attribute and loads each slide within the path we specified. We're assuming that every slide file name follows a slide-n.html syntax. Set these templates so that they're accessible to the controller and call the update method, which we'll write later.

```
angular.module('slidesDirective', [])
.directive('slides', function () {
  return {
    templateUrl: 'scripts/templates/slides.html',
```

Is native faster?

Yes, almost always. Fancy graphics are very appealing, but aren't necessary in making a good app. If you can design an app that does its job well, the average user won't notice that it's a web app unless you tell them.

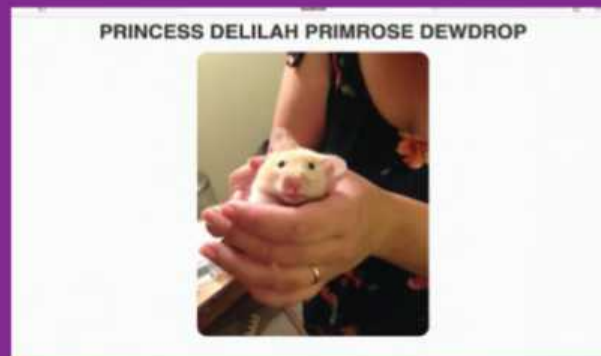
```
replace: true,
link: function ($scope, element, attributes)
{
  var templates = [],
  amount = parseInt(attributes.amount, 10);
  for (var i = 1; i <= amount; i++)
  templates.push(attributes.path + '/' +
  'slide-' + i + '.html');
  $scope.$parent.templates = templates;
  $scope.update(0);
}
};
});
```

13. Slide directive template

To enable smooth transitions, the DOM will have to hold at least three slides at any one time: the previous, current, and next. Otherwise, you'd get a jarring FOUC (Flash Of Unstyled Content) as the slide loads. This directive will automatically update when the slide object is updated thanks to ng-model.

```
<section class="slides" ng-model="slide">
<div class="slide slide-previous" ng-
```


Mobile Apps

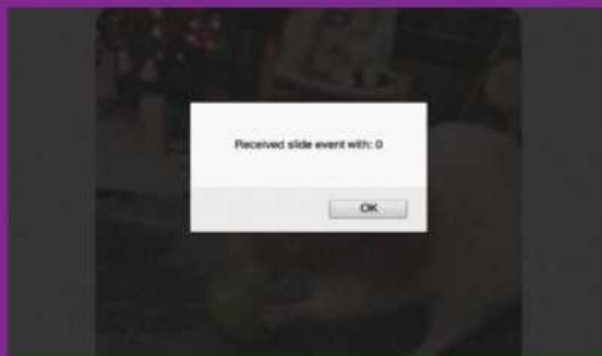


<Top Left>

• Finally we have a slide appearing thanks to our directive, there are only ever three slides in the DOM

<Top Right>

• Now that we have our keyboard events hooked up we can change the slide by pressing the arrow keys



<Bottom Left>

• We've received an event from the server, which triggers the 'changeSlide' method and updates the UI

<Bottom Right>

• The question dialog box will simply send the text inside that has been input here straight to the presenter

Example applications

The Socket.IO site (socket.io) has some impressive demos, including the ubiquitous chat app and the Pokémon game that is playable by everyone connected to the site called WePlay.

```
include="slide.previous">{{previous}}</div>
<div class="slide slide-current" ng-
include="slide.current">{{current}}</div>
<div class="slide slide-next" ng-
include="slide.next">{{next}}</div>
</section>
```

14. Stay within boundaries

We have to perform some fairly tedious boundary checking to make sure that we're not going to go over or under the slide's length, so before manually setting a value for previous or next values we'll pass them through these functions respectively to get a valid index.

```
$scope.setPreviousIndex = function (index) {
return index - 1 >= $scope.slides.length ?
index - 1 : 0;
};
$scope.setNextIndex = function (index) {
return index - 1 >= $scope.slides.length ?
index + 1 :
$scope.slides.length - 1;
};
```

15. Check index

We're doing a further check for the current index so it doesn't go to slides that don't exist. Nor do we want to press right and go beyond the amount of slides that we specified. If we do find ourselves at either extremity then we return the minimum or maximum index value.

```
$scope.setIndex = function (index) {
if (index >= 0 && index <= $scope.slides.
length - 1) {
return index;
}
if (index < 0) {
return 0;
} else {
return $scope.slides.length - 1;
}
};
```

16. Update method

The update method receives an index value for the current slide and simply updates the next, current and previous slides with the helper functions we've just written and contains them all within the \$scope.slide object. The templates object holds the HTML for each slide as a string and this is what's rendered in the slides directive.

```
$scope.update = function (index) {
if ($scope.slides) {
$scope.slide = {
next: $scope.slides[$scope.
```

```
setNextIndex($scope.index)],
current: $scope.slides[$scope.
setIndex($scope.currentIndex)],
previous: $scope.slides[$scope.
setPreviousIndex($scope.index)]
};
}
};
```

17. Determine the user

ChangeSlide receives a keypress event and we'll use this to work out which key was pressed in the next step. If this user isn't the presenter then we won't enable any keyboard control. If they are a presenter then use the 'emit' method on the socket manager to send the server the current slide's index.

```
$scope.currentIndex = 0;
$scope.changeSlide = function ($event) {
if (!$scope.isPresenter) {
return;
}
/* next step */
socket.emit('slide', $scope.currentIndex);
$scope.update($scope.currentIndex);
};
```

18. Control keyboard events

Sandwiched between deciding if the current user is a presenter and emitting the 'slide' event we check which way the slide should transition. We're also adding a way to quickly go to a slide between one and nine with the

final clause - each keyCode for one to nine is sequential (starting at 49 for 0).

```
var key = $event.which;
if (key === 39) {
  // right arrow key
  $scope.currentIndex = $scope.setIndex($scope.
    currentIndex + 1);
} else if (key === 37) {
  // left arrow key
  $scope.currentIndex = $scope.setIndex($scope.
    currentIndex - 1);
} else if (key >= 49 && key <= 57) {
  var desiredSlideIndex = key - 49;
  $scope.currentIndex = $scope.
    setIndex(desiredSlideIndex);
}
```

19. Change external slides

Still within our presentation controller we can listen to events from our server in a similar fashion to other JavaScript events by specifying the event name that we're going to listen for and a callback to execute. When we receive new slide information it'll be the index to go to.

```
socket.on('slide', function (index) {
  $scope.currentIndex = index;
  $scope.update(index);
  $scope.$apply();
});
```

20. Presenter listener

Likewise we'll listen to another event called isPresenter and set that value to whatever is applied (it'll either be true or false). We have to manually update the scope as when a Socket.IO event is fired it's independent of Angular so it doesn't know that the UI needs updating.

21. Listen to connections

Our client is now listening to all the right events but the server needs to control who sees which messages. We'll start filling in the rest of our server in events.js. When we receive a new connection we set up a listener that will listen to emits from that socket. We then relay the data received to other connected users with broadcast.

```
io.on('connection', function (socket) {
  socket.on('slide', function (data) {
```

```
socket.broadcast.emit('slide', data);
});
/* next step */
});
```

22. Set presenter

We need someone to present our presentation! This implementation is very rudimentary but the first person to connect will be the presenter. If there is not a current presenter then we set the next socket to connect as the presenter and let them know this by emitting the isPresenter event. Each socket is given an ID and it's important we store this ID for later.

```
if (!presenter) {
  presenter = socket.id;
  socket.broadcast.emit('slide', data);
  socket.emit('isPresenter', true);
  console.log(socket.id + ' is presenter');
}
```

23. Emit to a specific user

When the server receives a question we only want to tell the presenter about that question. We can access all of the current socket connections with `io.sockets.connected` and because we stored the presenter's socket ID, we specifically emit an event to them (in this case, the question event with the question data).

```
socket.on('question', function (question) {
  if (io.sockets.connected[presenter]) {
    io.sockets.connected[presenter].
      emit('question', question);
  }
});
```

24. Submit questions

Switching back to our presentation controller we'll add the functionality to submit and see these questions. We haven't got any validation checks on here, whatever is input into the textbook will be sent to the presenter. We up the currentIndex value by one because it's zero-based and don't want the presenter to always be one off!

25. Push notifications

When the presenter receives a question we'll display it as a notification on the top-right of their screen. We receive the slide's number and question so we'll do some quick microtemplating to format this data to be



<Above>

• Our last piece of functionality is complete, presenters can now see questions come through in real-time

Stream binary files

We've only transported plaintext but Socket.IO can also stream binary. This opens up many possibilities as it means you could stream files like audio or video.

slightly more readable. We then push this to an array that holds all of the notifications.

```
$scope.notifications = [];
socket.on('question', function (question) {
  var text = 'On slide {{slide}} someone asked:
    {{question}}'
    .replace('{{slide}}', question.slide)
    .replace('{{question}}', question.question);
  $scope.notifications.push(text);
  $scope.$apply();
});
```

26. Submit a question

We'll add a button to show to viewers of the presentation (but not the presenter) which, when clicked, will open a modal window with a text area for them to ask their question. By using ng-model we can reference this within our controller as \$scope.question.

27. Show notifications

Finally, as a presenter we'll list each of the question notifications with ng-repeat and show a close button so that the presenter can keep track of what has been answered. We haven't covered this functionality but it splices the notifications array; the full code for this tutorial is available on FileSilo.

```
<ul class="questions">
  <li data-ng-repeat="notification in
    notifications">
    {{notification}}
    <button type="button" aria-label="Close"
      data-ng-click="removeNotification($index)">x
    </button>
  </li>
</ul>
```

☞ We up the currentIndex value by one because it's zero-based and don't want the presenter to always be one off! ☞

Design and share a web app faster with React

Harness the power of your server to deliver an isomorphic web app to all of your users using Facebook React **tools | tech | trends** JavaScript, Node.js, React.js, D3.js, SVG



Over the last few years we've seen an explosion in the functionality that browsers provide. Gone are the days of simple text documents with a few images, now people expect beautiful, immersive

websites while also expecting it to be as fast as those text documents to load. This has led to a move to

'fat' clients where JavaScript is responsible for rendering much of the page. For modern, capable devices this is great however much of the time we're pandering to an elite (albeit growing) few who can afford such devices. Isomorphic JavaScript apps share code between the client and the server so you only write your application once but it's shared between both platforms. This means that a device that doesn't support

JavaScript can still see a JavaScript-generated chart from the server.

To explore this concept we're going to build a chart with D3.js that shows the average summer and winter temperature over 165 years. The chart will be pulled together with Facebook React which runs on Node and in browsers. We'll use Browserify to convert any Node-specific code to JavaScript for the browser.

1. Install dependencies

All of this power doesn't come cheaply, there are quite a few moving parts that we rely on, but first we need to install them all. Our Node app is going to run on Express, and compile and serve our React app. We'll use gulp and Browserify to convert this into code the browser can use. Create a package.json file with these dependencies and `$ npm install` them.

```
"dependencies": {
  "d3": "^3.5.5",
  "ejs": "^2.2.4",
  "express": "^4.11.2",
  "gulp": "^3.8.11",
  "gulp-browserify": "^0.5.1",
  "node-jsx": "^0.12.4",
  "react": "^0.12.2",
  "reactify": "^1.0.0",
}
```

2. Express server

Create a file called 'server.js', this'll bootstrap our server. If you've used Express before there shouldn't be any surprises here, we're serving static assets from a folder called 'public' and we're going to use EJS for our view engine. We're declaring our routes in a separate file which we'll create shortly.

```
var express = require('express'),
    path = require('path');
var app = express();
app.use(express.static(path.join(__dirname, 'public')));
app.set('views',
```

```
path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
require('./routes.js')(app);
app.listen(3000);
```

3. Install node-jsx

Next, within server.js also require node-jsx. JSX is an XML-like syntax for describing HTML and it feels more natural than building strings or creating DOM elements. Node-jsx looks for JSX files and compiles them from JSX to regular JS; so requiring is all you have to do, the rest is magic.

```
require('node-jsx').install();
```

4. Require dependencies

Create a file called 'routes.js'. This is where we will compile our React app and send it to the client. We will need to require React, the chart app which we are yet to write, and the data that the chart app will use. You can download this JSON file from bit.ly/1FOR3x9.

```
var React = require('react/addons');
var ChartApp = React.
  createFactory(require('./app/components/
    ChartApp').ChartApp);
var temperature = require('./app/data/
  temperature.json');
module.exports = function(app) {
  /* next step */
};
```

5. Render to string

This step is where the magic of server-side rendering

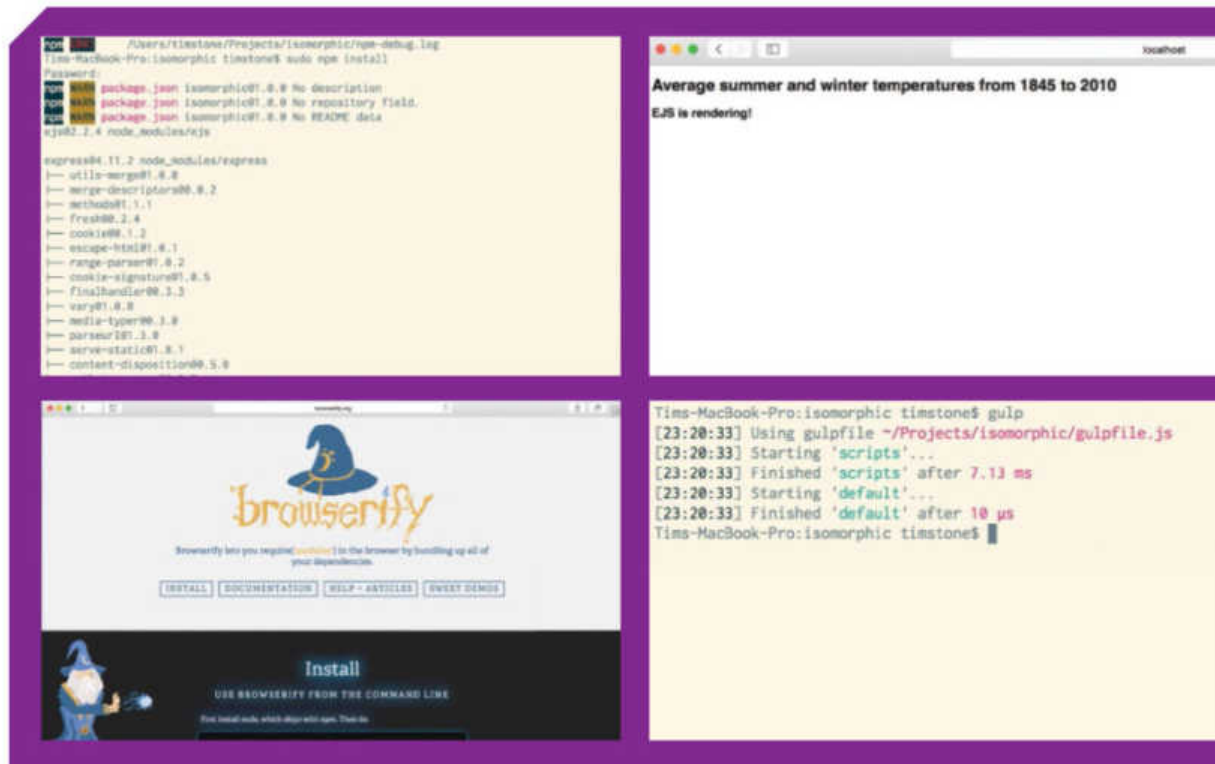
happens. React has a handy method called 'renderToString' which'll take all of the compiled JSX and render it to a string, ready for consumption by a browser. In one fell swoop we're instantiating our chart app with the temperature data and then converting it to a string and rendering it into the index file.

```
app.get('/', function(req, res){
  var compiledHTML = React.renderToString(
    ChartApp({
      data: temperature
    })
  );
  res.render('index.ejs', { chartApp:
    compiledHTML });
});
```

6. Create index.ejs

We're going to create a basic HTML page under a folder called 'views' and call it 'index.ejs'. The `<%->` is a placeholder for the HTML that we compile in the routes.js file and it'll output whatever we pass with a matching name. Note the 'react-main-mount', this is the node that the JS will look for.

```
<!DOCTYPE html>
<html>
<head>
<title>Temperature chart</title>
<link href="aestheti.css" rel="stylesheet"
  type="text/css">
</head>
<body>
```

<Top left>

• You can use '\$ npm init' to create a skeleton package. json file and then add these dependencies before '\$ npm install'

<Top right>

• Although this is not our final app, this small victory shown here tells us that EJS is rendering the page correctly

<Bottom left>

• Using Browserify means that exactly the same code and libraries will be used on the server and client

<Bottom right>

• Our small gulp file will pull all of these files together and combine them into a single file for the client

```
<h1>Average summer and winter temperatures
from 1845 to 2010</h1>
<div id="react-main-mount">
  <%- chartApp %>
</div>
<script src="main.js"></script>
</body>
</html>
```

7. Build the application structure

Before we go any further let's look at an overview of our app structure and explain how it interacts. So far we've written server.js, routes.js, package.json, and added index.ejs to views, but where's the code that's going to be shared? This will be ChartApp.js, the data, and main.js will glue it together.

```
app
components
ChartApp.js
data
main.js
package.json
public
aestheti.css
main.js [generated by gulp]
routes.js
server.js
views
```

8. Require modules

Main.js is fairly similar to what routes.js does but for the client in that it requires the same files and instantiates a

new React app. The crucial difference is that this time, instead of rendering out a string, we're mounting it to a DOM element. React on the client reuses the HTML outputted by the server.

```
var React = require('react/addons'),
    ChartApp = require('./components/ChartApp'),
    ChartApp,
    ReactApp = React.createFactory(ChartApp),
    mountNode = document.getElementById('react-
main-mount'),
    temperature = require('./data/temperature.
json');
```

9. Render app

Now create the app in a similar way except instead of rendering to a string we call React.render. Crucially we have access to the viewport's actual width and height so we can use that to set the chart to take up the available space. The second element is the node to attach it to ('#react-main-mount').

```
React.render(new ReactApp({
  data: temperature,
  height: window.innerHeight,
  width: window.innerWidth
}), mountNode);
```

10. Create a gulp file

For this client/server magic to happen we need an additional build step. This is in the form of Browserify and Reactify, which we'll run with gulp. Our gulp file will look at main.js, bundle up all of the required files and

Run gulp on save

You need to run gulp manually each time you change main.js or ChartApp.js. You could also hook gulp up so that it runs on save to prevent some frustration.

convert the JSX bits to JS. It then pipes all of this into a file also called main.js, which can be found under the public folder.

```
var gulp = require('gulp'),
    browserify = require('gulp-browserify');
gulp.task('scripts', function () {
  gulp.src(['app/main.js'])
    .pipe(browserify({
      debug: false,
      transform: [ 'reactify' ]
    }))
    .pipe(gulp.dest('./public/'));
});
gulp.task('default', ['scripts']);
```

11. Run gulp

You can then run this by navigating to the project folder in a CLI (like Terminal.app) and run \$ gulp. You could also run it through uglify which minifies all of the code as well as bundling it all into one file.

```
$ gulp
```

12. React Chart class

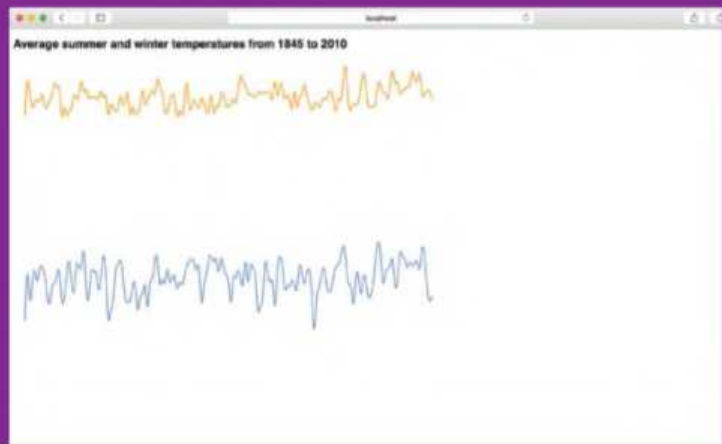
All of the code required to render our React app on the

Mobile Apps



<Above>

• D3 is the de facto charting library because of the power that it puts right into the hands of the creator



<Above>

• Here we can see the summer and winter series rendering. Without an axis this is what the server produces

server is in place with just one small hitch – there's nothing to render! Create a new file called 'ChartApp.js' under the components folder and we'll start to write our React app. Begin with the Chart, this is the wrapping SVG element.

```
var Chart = React.createClass({
  render: function() {
    var height = this.props.height + 20 + 30;
    var width = this.props.width + 50 + 80;

    return (
      <svg width={width} height={height}>
        <g transform="translate(20, 0)">
          {this.props.children}
        </g>
      </svg>
    );
  }
});
```

13. Use the React Line class

The Line class is fairly low-level and will output individual paths for our chart. We're also using a React method called `getDefaultProps`, this sets default properties if no property value is passed to it. A path in SVG will render a line going to the points specified in the `d` ('path descriptions') attribute.

```
var Line = React.createClass({
  getDefaultProps: function () {
    return {
      path: '',
      color: 'blue',
      width: 2
    };
  },
  render: function () {
    return (
```

```
<path d={this.props.path} stroke={this.
  props.color} strokeWidth={this.props.width}
  fill="none" />
);
}
});
```

14. React DataSeries class

The DataSeries class uses the Line class but we perform some D3 specific computations before setting the path data. The default interpolation we'll be using is 'basis', which smooths out extreme peaks leaving us with a nice-looking graph. If you're going for accuracy then 'linear' may be better.

```
var DataSeries = React.createClass({
  getDefaultProps: function() {
    return {
      title: '',
      data: [],
      interpolate: 'basis'
    };
  }, /* next step */
});
```

15. React DataSeries continued

The easiest way to communicate between React components with a parent-child relationship is through properties. In this case we're passing the colour of the series from here to the Line itself. We're also using D3 to create a SVG line for us and interpolating it so that each point joins to the next.

```
render: function() {
  var y = this.props.y,
      x = this.props.x,
      path = d3.svg.line()
        .interpolate(this.props.interpolate)
        .x(function(d) { return x(d.year); })
```

```
.y(function(d) { return y(d.temperature); });
return (
  <Line path={path(this.props.data)}
    color={this.props.color} />
);
}
```

16. Y axis class

The axes classes are slightly special cases because they require the DOM for D3 to create the required markup. To cater for this we will create the axis within the `componentDidMount` lifecycle method. This means that once the component has been mounted and has a node then the D3 axis method can be called.

```
var YAxis = React.createClass({
  componentDidMount: function () {
    var yAxis = d3.svg.axis().orient('left').
      scale(this.props.y);
    d3.select(this.getDOMNode()).call(yAxis);
  }, /* next step */
});
```

17. Y axis continued

The render function is fairly simple. We have a group element `<g>` and a text element beneath that. We're just applying some presentation properties to it so that it appears beside the y axis. 'text-anchor: end' is kind of similar to 'text-align: right', this'll ensure it appears at the top of the axis.

D3 applications

Use Nodemon (nodemon.io) as an easy way to automatically restart Node when you change a file so that you don't have to manually stop and start it.

```
render: function() {
  var textAnchor = {
    textAnchor: 'end'
  };
  return (
    /* next step */
    <g className="y axis">
      <text transform="rotate(-90)" y="6" dy="1em"
        style={textAnchor}>Temperature (°C)</text>
    </g>
  );
}
```

18. X axis class

The x axis has a very similar setup to the y axis class. The x axis additionally requires to know what the height of the chart is to position itself at the bottom. The '30' is to offset the height of its own element and could be done a little more elegantly than hardcoding a magic number.

```
var XAxis = React.createClass({
  componentDidMount: function () {
    var xAxis = d3.svg.axis().orient('bottom').
      scale(this.props.x);
    d3.select(this.getDOMNode()).call(xAxis);
  },
  render: function() {
    var transform = 'translate(0,' + (this.props.
      height - 30) + ')';
    return (
      <g className="xaxis" transform={transform} />
    );
  }
});
```

19. LineChart class

The LineChart class contains the majority of the chart logic. We start by setting some default property values. The year comes through as a number but we need to convert this to a date so that D3 can display it properly. We do this with a time-formatting helper from D3 itself.

20. Ordinal scale

Our data looks like [{"year": "1845", "Winter": "-2.5", "Summer": "-1"}, ...]. We want to split the winter and the summer temperatures into separate series. We're also using a D3 function which sets the scale to plot the axis and relationship between data points accurately. There are many different options for this but using the built-in scale functions greatly simplifies your code.

```
var summer = [];
var winter = [];
var scale = d3.scale.ordinal();
```

21. Create series

To split the raw data into two series we'll loop through it and push the summer values and year to an array called 'summer' and likewise for the winter array. We're also parsing the year as a date if it's necessary, but this only has to be done once per session.

Example

```
$ node server.js

Then navigate to http://localhost:3000 and click on the button to see some reactive events in action.
Try viewing the page source to ensure the HTML being sent from the server is already rendered (with
checksums to determine whether client-side rendering is necessary)

Here are the files involved:

App.js:

var React = require('react'),
    DOM = React.DOM, div = DOM.div, button = DOM.button, ul = DOM.ul, li = DOM.li;

// This is just a simple example of a component that can be rendered on both
// the server and browser

module.exports = React.createClass({

  // We initialize its state by using the "props" that were passed in when it
  // was first rendered. We also want the button to be disabled until the
```

Alternative approaches

There is no canonical way to handle server-side rendering. Another example is github.com/mhart/react-server-example which toggles the response based on what's being called, a page or a JavaScript file. It uses Browserify within the response to the client so that there isn't a separate build step, the tradeoff is that this is an expensive operation per-request. The best approach is to check out similar projects on GitHub and explore. In the finest examples of server-side rendering the user shouldn't notice the transition. The main difference will generally be that interactions become available. We've made ours quite obvious to demonstrate this but it could also be achieved with CSS.

```
data.forEach(function (d) {
  if (typeof d.year === 'string') {
    d.year = parseDate(d.year);
  }
  /* next step */
});
```

22. Combined data

The data comes to us as how much each month deviates from the average of that period across time. While this data could well be of interest it can be a little confusing, so we'll add the average temperature to how much the temperature deviates, eg summer 1920 has a value of -1.2 meaning it was 1.2 degrees cooler than the overall average of 15.2.

```
summer.push({
  year: d.year,
  temperature: 15.2 + parseFloat(d.Summer)
});
year: d.year,
temperature: 4 + parseFloat(d.Winter)
});
```

23. X scale

The x axis is a measure of time in years so we create a timescale with D3. Pass the 'domain', that is, the years to cover and the pixel range via the width property. Check out bit.ly/1GBW4wl for an intro to domain and range.

```
var x = d3.time.scale()
  .domain(d3.extent(data, function (d) {
    return d.year;
  }))
  .range([0, this.props.width]);
```

24. Y scale

The y axis is a linear scale and we pass the minimum temperature and maximum temperature to set the domain of the y axis. We specify the range in the same way but use the height instead of width as we're working vertically this time. This works on the

assumption that winter temperatures will always be lower than summer ones.

```
var y = d3.scale.linear()
  .domain([
    d3.min(winter, function (c) {
      return c.temperature;
    }),
    d3.max(summer, function (c) {
      return c.temperature;
    })
  )
  .range([this.props.height, 0]);
```

25. Tie it together

Finally tie it all together with each of our React classes. We're adding the wrapping SVG element, x and y axes, and the series. This will compile each of our React classes into a collection of SVG elements to represent our graph. You can only return one root node at a time otherwise it'll throw an error.

```
<Chart width={this.props.width} height={this.
  props.height}>
  <XAxis x={x} height={this.props.height} />
  <YAxis y={y} />
  <DataSeries data={summer} size={size} x={x}
    y={y} ref="Summer" color="orange" />
  <DataSeries data={winter} size={size} x={x}
    y={y} ref="Winter" color="cornflowerblue"
  />
</Chart>
);
```

26. Export module

We must not forget that this file is also a module. Of all the React classes that we've created the only part of it that we really care about is the overall chart, so we'll export the LineChart class which is used by routes.js and main.js. You could also split each of the classes into separate files.

```
module.exports.ChartApp = LineChart;
```

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Build a Chrome OS app with JavaScript

Use JavaScript to access nonstandard APIs and benefit from greater control in comparison to a typical web app





Chrome is only six years old, and yet, in such a short space of time (although a year is like an eon in technology) it's come to dominate browsing across our devices. On big screens and little

ones, Google has worked hard to make their browser the top dog in almost every conceivable space, and the low-cost PC market is not immune to this relentless march to dominance.

Chrome OS is an operating system based on the popular Chrome browser; JavaScript is a first-class citizen and the beauty of writing apps for Chrome operating system is that they will also work on almost any version of the Chrome browser.

It's said that the web is the platform, but as far as Google is concerned, Chrome is *this* platform and in this issue we're going to take that platform for a test-drive. Chrome OS has new, more powerful and further reaching JavaScript APIs than any other modern browser, and these APIs will also let us do a great deal more than a browser would normally be able to do.

We're going to create a minimalist stats board to teach us how to make Chrome OS apps and to demonstrate how far into the OS these APIs reach.

1. Get started

First we will need to download the project files from FileSilo. We're not going to write our own CSS or create our own icon files as this tutorial is about Chrome, not Photoshop! Unzip the project.zip folder and this will create a project that meets the minimum requirements for running a Chrome OS app.

2. The manifest.json

The manifest is where we define the prerequisites for the

app. Here, we give our app a title, paths to any icon files we may have, version number and any permissions our app may need. If we submit this app to the Chrome Web Store, the details used to describe and install the app will be taken from the manifest file.

```
{
  "name": "WDM Demo",
  "description": "A demo Chrome OS app for
  Web Designer Magazine.",
  "version": "0.1",
  "manifest_version": 2,
  "app": {
    "background": {
      "scripts": ["background.js"]
    }
  },
  "icons": { "16": "stats-16.png", "128":
    "stats-128.png" },
  "permissions": ["notifications", "system.
    cpu"]
}
```

3. Permissions

In our manifest, we define any permissions our app may need. Unlike a webpage, our users will not be asked for permission when we want to use a Web API (such as geolocation), instead, the act of installing the app is the act of consent that is required for us to act on the users behalf. For this app, we only want permission to use the Notifications API and the system.cpu API.

```
"permissions" : [
  "notifications",
  "system.cpu"
]
```

4. Background.js

In our manifest, we defined a background script, aptly named background.js. This file is where our app starts its life. Chrome OS will run this file before any other. In this file we've added a listener that will be triggered when the app has been started. Here, we tell Chrome OS that index.html is the one that will define our window and we will then pass bounds to set the size of that window once it's been opened.

```
chrome.app.runtime.onLaunched.
addListener(function() {
  chrome.app.window.create('index.html', {
    'bounds': {
      'width': 1024,
      'height': 300
    }
  });
});
```

5. Let's experiment

None of the APIs that we're using in this tutorial are considered 'experimental' but if want to get creative with Chrome OS apps, then enable the experimental APIs. Just enter 'chrome://flags' in Chrome's URL bar, search for 'Experimental Extension APIs' and click enable.

Experimental APIs and you

Google likes to use Chrome OS as its testing ground for experimental APIs. Long before anything gets submitted to the W3C (if ever) Google will have written and redeployed an API a thousand times.

Extensions

Load unpacked extension...

Pack extension...



WDM Demo 0.1

A demo Chrome OS app for Web Designer Magazine.

Permissions Launch Reload (⌘R) Errors

ID: cjokfenjijnbdhnbdfpcfidcobpihe

Loaded from: ~/Documents/Articles/Chrome App/public

Inspect views: background page (Inactive)

☐ Allow in incognito ☒ Collect errors

idle	kernel
10190514	100718
idle	kernel
16017444	004419
idle	kernel
10192736	106675
idle	kernel
10020432	004425
idle	kernel
10192047	100158



<Left>

- You can load your app into Chrome without having to go through a packaging process. Just click 'Load Unpackaged Extension' and navigate to your project folder with the manifest.json in it

<Top left>

- To the untrained eye, these numbers will mean very little, but they represent the time the systems processors are taking to complete certain system or user ran processes

<Top right>

- Chrome's notifications API is separate from the Web Notifications API which ties into the OS notifications protocol. Still, they do the job rather well

Mobile Apps

6. Install our app

Before we can run our app, we need to install it. Don't worry, this isn't as daunting as it sounds. Open Chrome and click on the hamburger menu to the right of the URL bar, find more tools and then click 'Extensions' (this may vary across different versions of Chrome). Tick 'developer mode' at the top of the page and then click 'load unpacked extension' and select the folder with our project in it.

7. Run our app

Our app will pop up in the list of extensions, but we can't run it from here. If we want to run our newly loaded app (from Chrome OS) we will need to head to the apps icon in your taskbar and select it from there or (from Chrome) open a new tab, click the Apps button in the top left of the window and run it from there.

8. First run

On our first run, our app is pretty empty looking, because, well, it's empty, but it won't be empty for too long. Just open up index.html for editing and add the following code:

```
<html>
<head>
<meta content="text/html; charset=utf-8"
```

```
http-equiv="Content-Type">
<title>WDM || Notes</title>
<link rel="stylesheet" href="styles.css"
type="text/css" />
<meta name="viewport" content="initial-
scale=1.0, user-scalable=no" />
</head>
<body>
<div id="meters"></div>
<script src="scripts/core.js"></script>
</body>
</html>
```

If you restart the app, you'll see that not much has changed, this is because we're going to use JS to create our DOM for us.

9. A functioning app

Open scripts/core.js for editing, all of the code is already there to be run, but we're not initialising it yet. Go to the bottom of the file and uncomment the line '//__chrome_cpu_watcher.init();' and then restart the app. Restart the app, voila! we have a fully functioning CPU meter with Chrome's CPU API. Let's walk through how this all works.

```
(function(){
//__chrome_cpu_watcher.init();
})();
```

10. The chrome window object

Whenever we want to interact with an API that is nonstandard, that is, Chrome OS/Chrome app specific, we do so with the global 'chrome' object. The APIs that are available are determined by the permissions argument set in our manifest file, but we can manipulate

the window of our app. On line 111 of core.js, we do just that by setting the width of our window to be large enough that it can accommodate the number of processors our Chromebook or PC has.

11. System.CPU

One of those nonstandard APIs is the system.cpu API and this API enables us to get real-time information about the state of our Chromebook's processor (or processors). This API has only one method, getInfo(), and we use its callback to set of our app on line 105. We need to know about the system we're analysing before we analyse it.

12. Create the CPU meters

Each different system has a different number of processing cores and the system uses the CPU differently. So, we need one meter per core and we need one extra one for the total. We create these meters with the createMeters() function between line 8 and 47 in core.js and we call that on line 109 inside of our first call to chrome.system.cpu.getInfo() on line 105. We're creating the HTML with a document fragment, but the HTML is equivalent to:

```
<div class="meter">
<h1>1</h1>

<p class="percentage">20%</p>
</div>
```

13. Get Chrome CPU data

So, now that we have meters, we need to get them moving! Unless you have a superpowerful Chrome OS device, those meters should never be stationary for long. In order to work out the percentage of our system usage, we need to do a little math over time, so we'll call 'chrome.system.cpu.getInfo(updateMeters);' on an interval of 500ms on line 114.

14. The CPU data

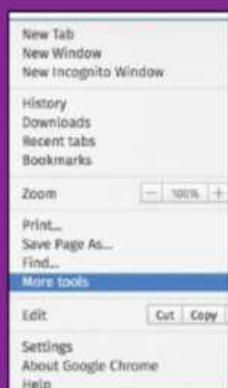
Getting our CPU data is really easy, but making sense of it is a little tough if you're not a sys dev. These are the 'times' that are taken up on the processors by different processes.

We can't do much with it as it is, in order to calculate a percentage, we need to compare two bits of data sampled over time.

```
"processors": [
{
"usage": {
"idle": 17674141,
"kernel": 1859709,
"total": 23680054,
"user": 4146204
}
},
{
"usage": {
```

Other platforms

If writing an app for one platform isn't your cup of tea, you can deploy Chrome apps to iOS and Android too. But be aware that some APIs might not be supported straight out of the box.



<Top left>

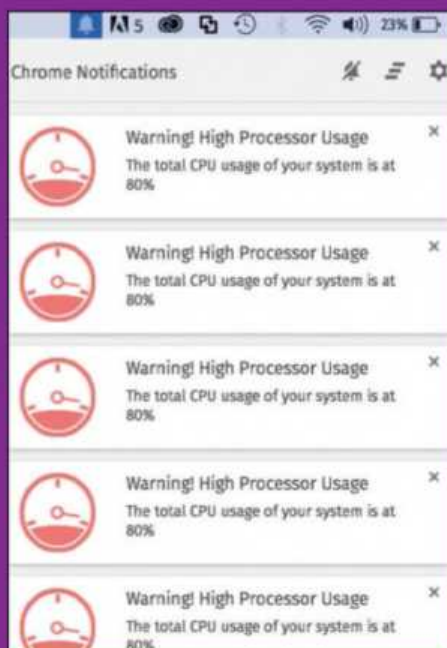
- The Extensions menu is buried quite deep in Chrome's hamburger menu, but you can find it under 'More tools'

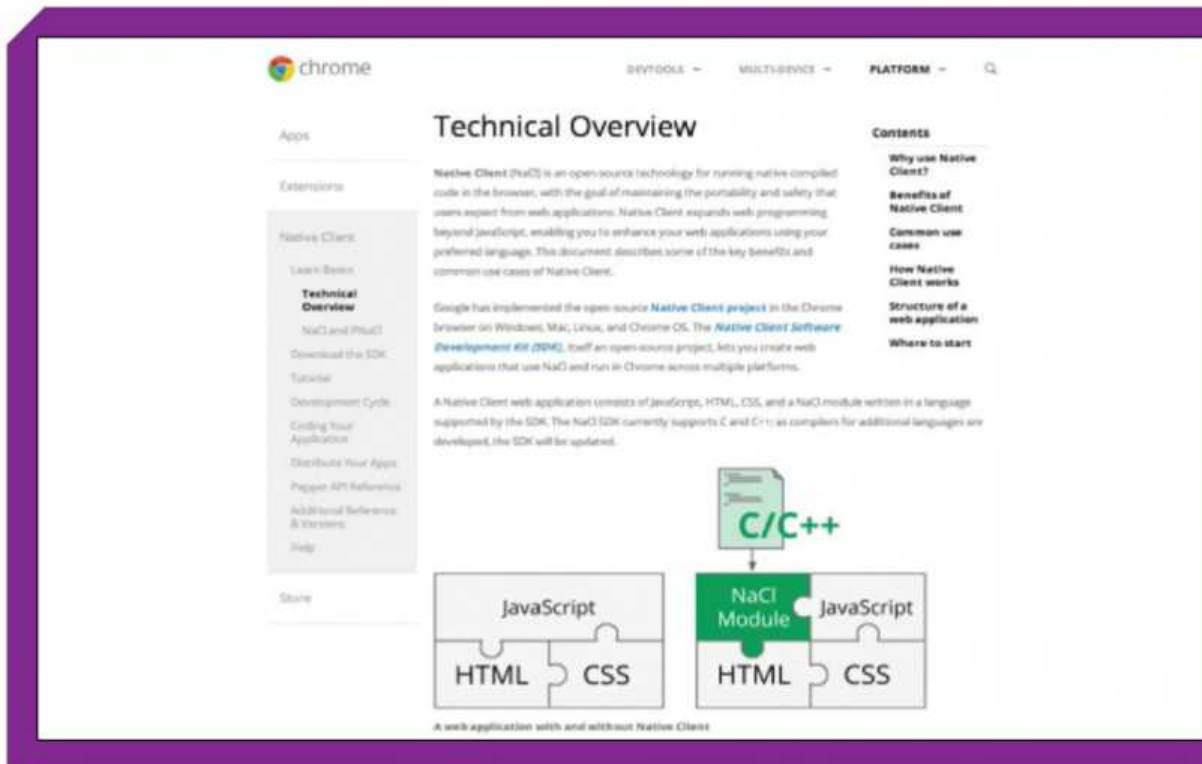
<Top right>

- You can run the app from the Extensions window, but once it's installed you can run it from your apps window

<Right>

- Chrome has its own notifications centre. On OS X you can find it in the menu bar, it's a little bell icon





The native client

Google wants you to write apps in JavaScript, that's why we have all of these fantastic, low-level APIs that let us do almost anything we can conceive of these days. That doesn't mean, however, that JavaScript is the only way of getting things to run in Chrome - there's also the native client. If you really want to create something that is mind-blowingly fast, you can run compiled C/C++ code in Chrome using a technology called PNaCl (Pinnacle). These extensions run much the same as their JavaScript equivalents, but they are platform dependant, so you still need a version for Windows/Mac/Linux which is a bit of a downside - but you have full control of the extensions capabilities. The question you have to ask is "Do I really need all of this power and control?". It's up to you!

```
"idle":23251012,
"kernel":245406,
"total":23679093,
"user":182675
}
}
]
```

15. The first instance

The first time we get our CPU data into `updateMeters`, we can't do anything with it, so we will skip over the calculations that are on lines 56 - 96 and put it straight into our `previousData` variable at the top of our function.

16. Compare the data

The second, and every subsequent time we get processor data, we can compare it with the data from the previous time `updateMeters()` was called. On line 62, we turn those big looking numbers from Step 12 into a friendly percentage out of 100. On line 63, we turn our percentage into an angle that we want the needle on our meters to move by, which we do will do so with `transform: rotate()`.

```
var used = Math.floor((processors[b].usage.
```

```
kernel + processors[b].usage.user -
previousData.processors[b].usage.kernel -
previousData.processors[b].usage.user) /
(processors[b].usage.total - previousData.
processors[b].usage.total) * 100),
indicatorRotation = (220 / 100) * used;
```

17. UsageTotal

For each core we will now add the percentage that is being used to `usageTotal`. When we've added up all of the processor usage, we can then divide it by the number of processors to get the total CPU usage of our system out of 100 per cent. We will do this on lines 72-75 and this is why we added 1 to the `numberOfProcessors` on line 109, it's so that we had an extra meter for our overall usage.

```
var totalUsage = (((usageTotal /
(numberOfProcessors * 100)) * 100) | 0);
meters[meters.length - 1].getElementsByClassName('percentage')[0].textContent =
totalUsage + "%/100"
meters[meters.length - 1].
getElementsByTagName('img')[0].style.
transform = "rotate(" + (220 / 100) *
totalUsage + "deg)";
```

18. Exceeding usage

Now, we have all of the code in place to visualise our CPU, if we were to run the app at this point, we'd see our CPU meters happily bobbing up and down as we used our Chromebooks, but we can't sit here watching these meters all day, we need something that will tell us if we're in danger, like say, if our total CPU usage exceeds 80 per cent. That's where the Notifications API comes in.

19. Red alert

Triggering a notification with Chrome OS is really quite simple, there are multiple types, but all we're going to use is the 'basic' type. This type includes an icon, a title and a message - all we need. On lines 77-94, we monitor the total CPU usage, if it goes above 80 per cent, we trigger a notification and we won't trigger another warning until it's dropped beneath 80.

```
chrome.notifications.create({
type : "basic",
title : "Warning! High Processor Usage",
message : "The total CPU usage of your
system is at " + totalUsage + "%",
iconUrl : "stats-notification.png"
}, function(e){});
```

20. Round up

That's it. We've put together a really simple Chrome OS app that monitors and warns us of excessive CPU usage, but we've only scratched the surface here of what Chrome OS APIs can do - with APIs like Serial or Bluetooth, you could transfer files wirelessly or even use them to drive a 3D printer!

🧠 We've put together a really simple Chrome OS app that monitors and warns us of excessive CPU usage 🧠

 **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Build a mobile web app with NativeScript's library

Use NativeScript to make apps for iOS, Android and Windows Phone and redefine mobile programming **tools | tech | trends** JavaScript, Telerik Platform, Telerik companion app



itobi's PhoneGap redefined the world of mobile app development by bundling web content with a browser stub.

NativeScript goes one step further: its developers provide a custom JavaScript Runtime which can access native operating system classes without cumbersome

plugins. A separate set of abstraction classes provide a common interface, sparing developers the effort of maintaining different code bases for each available operating system.

NativeScript's approach is beneficial in that it permits you to use native controls directly: applications rendered in WebViews tend to look out of place. Furthermore, it simplifies the creation of custom logic for

developers who are unfamiliar with Java and/or Objective C.

Even though Telerik has not yet released a final version of NativeScript, the open source product already shows significant progress and is more than deserving of your attention. A live licence is expected any time soon – so you better get coding while the coding is good!

1. Create an app

Sign up at telerik.com/nativescript first. Once logged in, click Create app to create a skeleton container. Click 'Create AppBuilder Native project' to start working with NativeScript and select NativeScript Blank (JavaScript).

2. Analyse the IDE

Telerik Platform will then create the project skeleton. Running the app on a device requires you to download the companion app (Telerik NativeScript) to your phone: it is available in the iOS App Store, on Google Play and in the Microsoft Store. Click Build and follow the instructions.

3. Inflatable GUIs

Now that the toolchain is in place, we can start to create a user interface declaration. Even though NativeScript permits the direct use of native control classes, using interpreted XML layouts tends to yield faster results. Double-click main-page.xml, and replace its contents with the code accompanying this step.

```
<Page xmlns="http://www.nativescript.org/tns.xsd" >
  <StackLayout>
    <Label text="R:" />
    <TextField id="TxtR" width="200"/>
    <Label text="U:" />
    <TextField id="TxtU" width="200"/>
    <Label text="I:" />
    <TextField id="TxtI" width="200"/>
    <Button id="CmdGo" text="Go" width="100"/>
  </StackLayout>
```

 </Page>

4. Add an event handler

User interaction makes widgets emit events. Button clicks should cause a recalculation of the values shown in the fields. This can be done by adding an event handler to the button. Since the button control exposes a property, we can integrate our new event handler by changing a single line in the XML file.

```
<Page xmlns="http://www.nativescript.org/tns.xsd" >
  <StackLayout>
    . . .
    <Button id="CmdGo" tap="buttonTap" text="Go"
      width="100"/>
  </StackLayout>
</Page>
```

5. Export the logic

Accompanying logic is contained in main-page.js. The most basic version of an event handler consists of the declaration of a method body, which is then added to the exports object. The NativeScript runtime will not be able to find the event handler if it is not part of the exports.

```
var view = require("ui/core/view");
function buttonTap(args) {
}
exports.buttonTap = buttonTap;
```

6. Find our widgets

The args-object provides a group of members which can

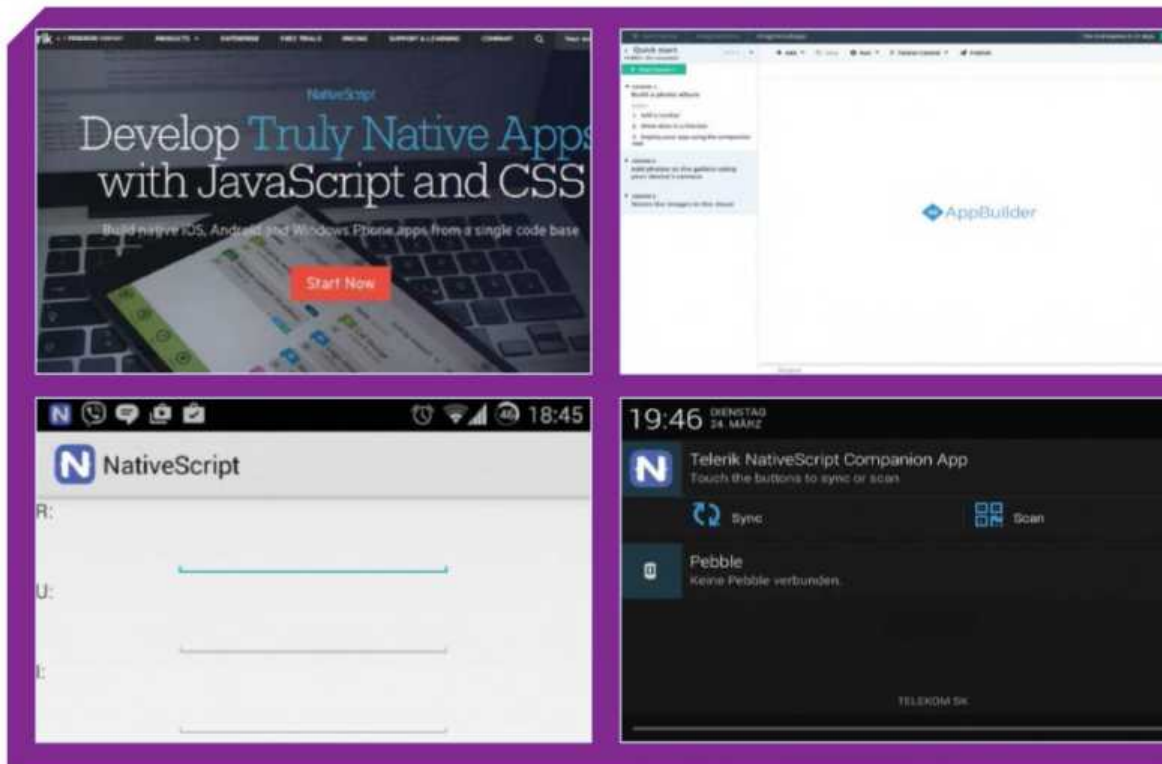
be used for traversing the object tree resuscitated from the XML declaration. The snippet demonstrates the traversal procedure for the GUI – we pick out the three labels, which come as children of the parent of the button object.

```
function buttonTap(args) {
  var sender = args.object;
  var parent = sender.parent;
  if (
    parent)
  {
    var txtR = view.getViewById(parent, "TxtR");
    var txtU = view.getViewById(parent, "TxtU");
    var txtI = view.getViewById(parent, "TxtI");
  }
}
```

7. Get calculating

Creating the actual logic is not particularly difficult. Start by analysing the contents of the text fields shown. If the R field is empty, it is reasonable to expect that the user has provided values for voltage and current – practical applications should perform more stringent checking of their input values.

```
function buttonTap(args) {
  var sender = args.object;
  var parent = sender.parent;
  if (parent)
  {
    . . .
    if(txtR.text.length==0)
```

<Top left>

• Click the big red button to get started, you can sign up to NativeScript or log in if you already have an existing account

<Top right>

• The NativeScript IDE is ready and waiting - just click a file on the right in order to open it up for editing

<Bottom left>

• Thanks to the abstraction classes, our XML file gets inflated into a user interface which looks almost native

<Bottom right>

• The Android OS, for example, will hide the buttons if the notification is not positioned at the top of the event list

```
{
  txtR.text=parseInt(txtU.text)/parseInt(txtI.
  text);
}
}
```

8. Sync the code

Telerik's online IDE stores changes automatically. Pushing changes to your Android phone is really easy: expand the notification, and click the Sync button. The NativeScript companion app will then proceed to downloading and committing any changes deemed important. Your phone does not need to be connected to a desktop for the downloads.

option, a promise object is returned. We use its 'then' method to schedule a function for delayed execution. See:

```
var dialogs = require("ui/dialogs");
dialogs.confirm({
  . . .
}).then(function (result) {

  if(result==true) //OK
  {
    txtR.text=parseInt(txtU.text)/parseInt(txtI.
    text);
  }
});
```

9. Complain about errors

NativeScript's libraries provide a wide variety of constructor objects which permit you to access their features. Our snippet obtains a reference to the dialog's object.

```
var dialogs = require("ui/dialogs");
dialogs.confirm({
  title: "Error!",
  message: "You must empty the R field",
  okButtonText: "Calculate anyways",
  cancelButtonText: "Sorry!"
});
```

10. False promises

Our message box runs its magic in an asynchronous fashion. Since blocking the GUI thread is not an

11. Add another page

With that, work on the calculator page is all but done - it is now time to add a second page to the application. Click Add New file, and enter 'tampage.js' into the file name field. Then, create a form definition by repeating the process with the file name tampage.xml.

12. Initial page change!

It's time to change the page shown after application start-up. This can be accomplished by opening the app.js file which contains the declaration of the application object. Next, change the value of the mainModule property to match the newly created group of files.

```
var application = require("application");
application.mainModule =
```

Tell me more!

The official NativeScript documentation is stored on GitHub. Visit it at [agithub.com/NativeScript/docs](https://github.com/NativeScript/docs), and scroll to the bottom of the page in order to start reading.

Yarr, kick a pirate!

NativeScript is interpreted at runtime. It will only be a question of time until attackers figure out a way to extract code from apps. Obfuscating code before compilation can help though...

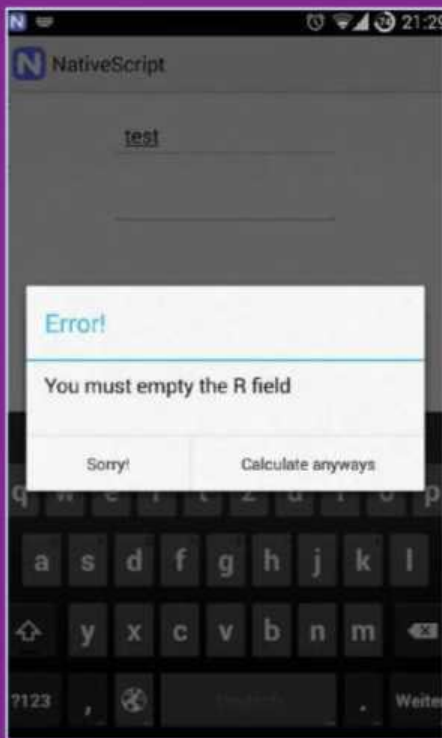
```
"app/tampage";
application.start();
```

13. Operating system events

The application class provides a group of members which can be populated with custom logic for handling operating system events. The code on FileSilo shows the possible fields along with basic handlers illustrating the usage of the provided parameters. Handling the events is not mandatory.

14. Dynamic event handling

Tampage is currently empty. Replace it with the code accompanying this step. The button is now declared without an inline event handler. Instead, the page is provided with a method to be invoked once the inflation is done.



<Above>

- Our MessageBox is displayed if the user enters useless information

```
<Page loaded="loadStartView">
<StackLayout>
<Button id="calcButton" text="Calculate a
resistance" width="600"/>
</StackLayout>
</Page>
```

15. Page switching

With that, it's time to switch the page in response to the button being pressed. OnLoad is responsible for setting the event handler as the form is loaded. The actual page switch is performed in the inline function provided.

```
var view = require("ui/core/view");
function onLoad(args) {
var page = args.object;
var my = view.getViewById(page,
"calcButton"); my.on("tap", function () {
var frames=require("ui/frame");
frames.topmost().navigate("app/main-page");
```

It's just JavaScript

Deploying large amounts of native code is a sure-fire way to reduce portability. Just check to see whether a JavaScript solution for your problem exists if you encounter a situation that is not already covered by the NativeScript libraries.

```
});
}
exports.loadStartView = onLoad;
```

16. Data binding

In the following steps, our example will receive a small list which is to be populated from an array object. Start by adding the specified bit of markup to the tamspace XML file:

```
<Page loaded="loadStartView">
<StackLayout>
<ListView id="myList" loadMoreItems="listVie
wLoadMoreItems" />
<Button id="calcButton" text="Calculate a
resistance"
width="600"/>
</StackLayout>
</Page>
```

17. Create an observableArray

Data shown in lists has the unpleasant habit of changing from time to time. NativeScript can handle updates automagically if the underlying data storage is derived from the observableArray class. Load it via require, and use it as a wrapper around a set of different electrical components.

```
var observableArray = require("data/
observable-array");
var labelModule = require("ui/label");
function onLoad(args) {
var page = args.object;
var myList = view.getViewById(page,
"myList"); var items = new
observableArray.ObservableArray(["Resistor",
"Diode", "Rheostat"]);
myList.items = items;
```

18. Create some elements

ListsViews invokes the itemLoading callback when new items need creating. It then receives an args.view parameter to be populated with the object(s) in the slot allocated. Now populate view with a simple label.

```
myList.on("itemLoading", function (args) {
if (!args.view) {
args.view = new labelModule.Label();
}
args.view.text = items.
getItem(args.index);
});
```

19. Annoy the list

With that, it's time to modify the underlying storage. This is accomplished via a method which adds a value to the array. As the window object does not exist in NativeScript, Telerik chose to provide the set XXX methods via a helper object.

```
var timer = require("timer");
function onLoad(args) {
var page = args.object;
var myList = view.getViewById(page,
"myList"); var items = new observableArray.
ObservableArray(["Resistor", "Diode",
"Rheostat"]);
myList.items = items;
myList.on("itemLoading", function (args) {
if (!args.view) {
args.view = new labelModule.Label();
}
args.view.text = items.getItem(args.index);
});
timer.setInterval(function () {
items.push("XXX");
}, 3000);
```

20. GUI to code

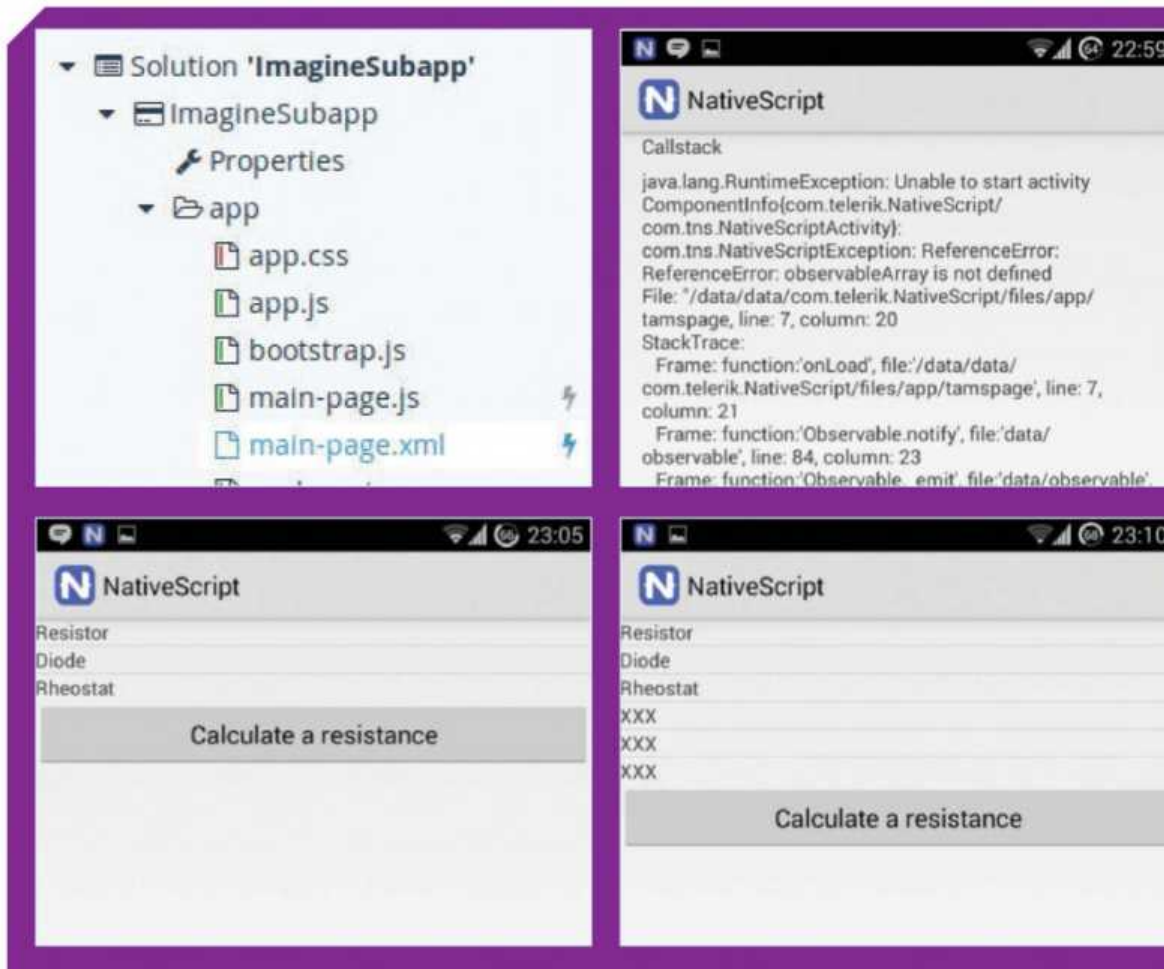
Our resistance calculator uses object references to retrieve the values needed for the computation. It would be much nicer if the updating was handled by the framework. We can accomplish this by creating a model class which is then bound to the form.

```
var dataField = {
valu: "24",
vali: "35",
valr: ""
};
function assignModel(args)
{
var page = args.object;
page.bindingContext = dataField;
}
function buttonTap(args)
{
var sender = args.object;
var parent = sender.parent;
if (parent)
{
if(dataField.valr.length==0)
{
dataField.valr=parseInt(dataField.valu)/
parseInt(dataField.vali);
}
else
```

21. Bind to the model

Setting the loaded property ensures that the initialisation code gets invoked as the app starts up. The text fields are bound to the individual fields of the model via binding strings. A double { informs the parser that a binding command is to follow, it is also possible to set up bindings entirely from code:

```
<Page xmlns="http://www.nativescript.org/
tns.xsd"
loaded="assignModelToView">
<StackLayout>
```



<Top left>

- Tampage.js lives in the /app/ subfolder. This affects the string which is to be passed to the page-changing function

<Top right>

- Providing invalid code or XML leads to an error such as the one shown in the figure screenshot above

<Bottom left>

- Our list populates itself automatically using the values stored in the object that has been passed in as the model

<Bottom right>

- Changes to the model's content are easily viewable as they are reflected in the user interface automatically

```
<Label text="R:" />
<TextField id="TxtR" width="200"
text="{{valr}}" />
<Label text="U:" />
<TextField id="TxtU" width="200"
text="{{valu}}" />
<Label text="I:" />
<TextField id="TxtI" width="200"
text="{{vali}}" />
```

22. Make it observable

The code from this step does not update the text box contents. JavaScript does not provide a way to notify objects about property changes, NativeScript addresses this by introducing a set of observable classes. Data models are based on this class with extra wiring, put to good use in our engine

```
var observable = require("data/observable");
var dataField = new observable.Observable();
dataField.valu = "24";
dataField.vali = "35";
dataField.valr = "";
function buttonTap(args) {
    . . .
    if (parent)
```

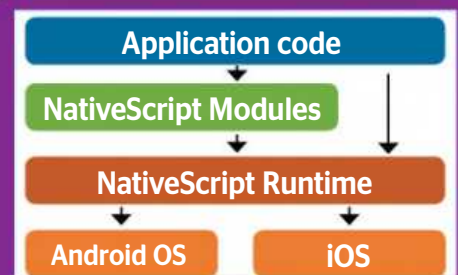
```
{
    if(dataField.valr.length==0)
    {
        dataField.set("valr",parseInt(dataField.
valu)/ parseInt(dataField.vali));
    }
}
```

23. Swipe to go back

Most NativeScript controls provide a predefined listener object, it can notify your application of incoming events. We will use this feature to provide our users with a convenient way to get back to the main page of the application. The code is relatively simple: a listener is assigned to the page. Args contain further information about the gesture.

21. Styling

Telerik's abstracted controls can be modified to suit your liking by adding CSS declarations to the application. The runtime will try to apply the specified design to the native controls as best as it can. Since the actual rendering is done by the operating system, one-to-one coverage can not always be achieved. More information on that is available via docs.nativescript.org/styling.html.



Release my app?

Click the Publish button to open the export wizard shown in the figure. As of this writing, Telerik does not permit the global distribution of NativeScript applications. Instead, developers can package their app into Telerik's AppManager, thereby providing access to a select group of customers.

As of this writing, the company has not yet decided when applications will be permitted to be released to the general public via app stores. However, we can reasonably assume that the process will be similar to the one used by completely native apps: Telerik's website will emit an APK or IPK file, which is to be uploaded into the app store of the vendor of choice. Users will not notice that they are dealing with a NativeScript application as the runtime is likely to be bundled.

↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Code a PhoneGap memo app with photos

Pass files onto your phone quickly and make them run native with PhoneGap Desktop **tools|tech|trends** PhoneGap Desktop, PhoneGap Developer, Brackets



Recently the PhoneGap team created an app for phones that could pair with the desktop using command-line tools to sync the development environment together with a

live testing environment on the phone. This enables designers and developers to see instant

refreshes of their app on the device so that code can be tested immediately without having to endure a lengthy build process.

So what's so special about this tool? Firstly, testing phone-specific features, like using the camera, especially if you are a PhoneGap Build user, took some time to get to the device. Now when you save a file such as HTML or CSS, the phone automatically

reloads the content and you can test your app as soon as it's transferred. Secondly, this is a simple user interface that enables you to manage multiple PhoneGap projects on your computer, making development much easier. Thirdly, iOS developers can get a taste of their app on the device without having to pay out for a developer licence until they are ready to start building.

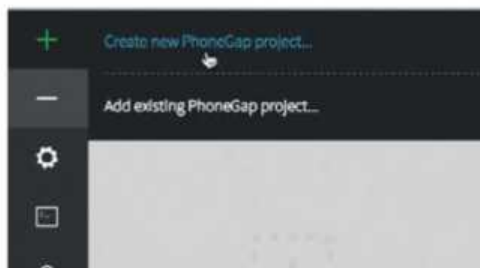
1. Get started

First step is to get the PhoneGap Desktop App for the platform of your choice. Visit phonegap.com/blog/2014/12/11/phonegap-desktop-app-beta and under 'Get the app' click on either Windows or Mac to get your version of the app. After it has downloaded follow normal install procedures for your computer.



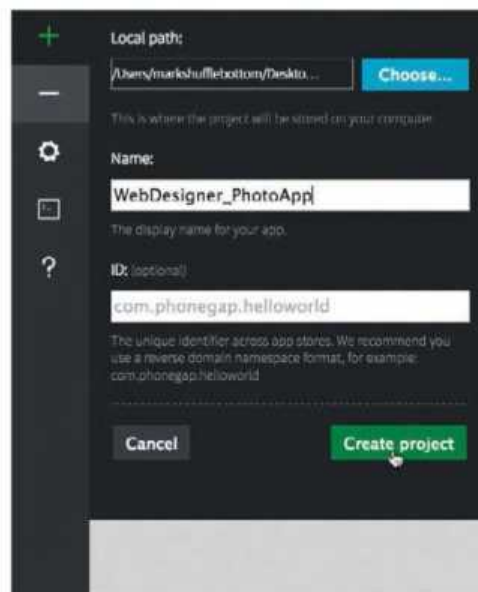
2. Open the app

After installation, the app is usually just extracted, so move it into your Applications folder and then double-click it to start. The interface is extremely minimal because there is not a lot of options for the app. Click the plus symbol to create an app and click 'Create new PhoneGap project...'



3. Name the app

Click on the Choose button and browse to the folder that you want to save the PhoneGap App to. Once you've done this click OK and then add a name for the app. It's not necessary at this stage as it can be added later, but you can add a reverse domain name for the app. 'Click Create Project' to continue.



4. Get the Developer App

You have created your first project and you need to see this on your phone, so go to your respective phone's app store and search for 'PhoneGap Developer App'.

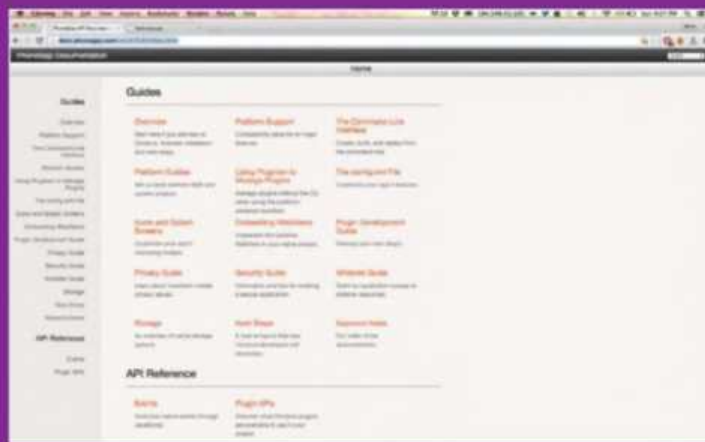
Download this app to your device. Once downloaded you can start this app on your device. When this is running you will see that you are asked to put in the server address.



5. Connect the two together

On the desktop app there is a green bar at the bottom, this tells you the address that the server is running at. You will need to be on the same Wi-Fi network as your phone. Put the address shown into your phone and hit Connect. On the screen you will see a green flashing message stating 'device is ready'.





<Above>

• Obviously as the main code is accessing a phone, PhoneGap (also known as Cordova) is used to access the native phone features through JavaScript. The documentation can be found at docs.phonegap.com/en/4.0.0/index.html

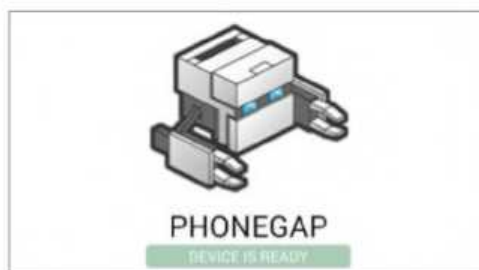


<Above>

• In the tutorial it is necessary to add and remove classes to control animation and to manipulate text in the document (see Step 8). For this the jQuery library is being used so that this can be done relatively easily

6. Your app on the phone

What you are seeing on your screen is the HTML, CSS and JS from the demo app being placed on your phone as an app. From the tutorial files on Filesilo, copy the WWW folder from the Start folder over the existing WWW folder. This just has the font, CSS and HTML in place so that we can create the app with JavaScript.



7. Create the app

Creating the actual logic is not particularly difficult. Start by analysing the contents of the text fields shown. If the R field is empty, it is reasonable to expect that the user has provided values for voltage and current - practical applications should perform more stringent checking of their input values.

```
var overlay=false;
var memos = [];
var key =
"memos";
var pic;
```

8. Detect user input

When the user presses the 'new' button in the HTML, this checks to make sure the overlay effect has not been set, the 'add' div section is made visible and set to animate in with a bounce from the left. As this is now out, the rest of the interface is hidden and therefore the overlay is true.

```
$( "#new" ).click(function() {
if (overlay == false){
$('#add').removeClass('hidden');
$('#add').addClass('animated
bounceInLeft');
overlay = true;
}
});
```

9. Other button inputs

When the user clicks on the submit button they are going to create a memo so here, the appropriate createMemo function is called. In the case of pressing the photo button the accessCam function will be called to start using the phone's camera. In both cases the code for these functions hasn't been added yet.

```
$( "#submit" ).click(function() {
createMemo();
});
$( "#photo" ).click(function() {
accessCam();
});
```

10. Access the phone's camera

In this function the camera on the phone is called and in most phones, it will bring up the native phone's camera app. Here the quality of the finished JPEG is set at 75 quality and the target width and height is set at 100 pixels so that it will fit within the interface that has been created in the HTML.

```
function accessCam(event) {
navigator.camera.getPicture(onSuccess,
onFail, {
quality: 75,
destinationType: Camera.DestinationType.
DATA_URL,
```

Desktop to mobile

When you have the PhoneGap Desktop App open it launches a server in the background, powered by Node.JS. This server sends the files to the phone as if they are a native app on the phone.

```
targetWidth: 100,
targetHeight: 100
});
}
```

11. Successful photo capture

With that, work on the calculator page is all but done - it is now time to add a second page to the application. Click Add New file, and enter 'tampage.js' into the file name field. Then, create a form definition by repeating the process with the file name tampage.xml.

```
function onSuccess(imageData) {
pic = imageData;

$('#preview').prepend(' pipe.position.x &&

memos.push(memo);
```

14. Store the data

The storeMemos function is called which will write the memo into the user's local storage as JSON. The interface needs to be updated and the overlay removed. The appropriate classes are removed and then the animation for moving the overlay out is added. A listener checks that this finishes, and then the overlay is hidden and the variable set to false

```
storeMemos();
$("#add").removeClass('animated
bounceInLeft').addClass('animated
bounceOutLeft');
$("#add").one('webkitAnimationEnd
mozAnimationEnd MSAnimationEnd oanimationend
animationend', function() {
  $("#add").removeClass('animated
  bounceOutLeft').addClass('hidden');
  overlay = false;
  return false;
});
```

15. Clean up the input

As the overlay interface is animated off the screen the memo needs to be added to the page, so the appropriate function is called. The text field is cleared of the current memo so if the user wants to add another memo they see an empty text input area and the image preview is removed as well.

```
addMemoToPage(memo);
$("#memo").val('');
$("#preview").empty();
}
```

16. Add to the page

The function to add the memo to the page is being created now and here the memo's element in the HTML is grabbed and a new list element is added within that unordered list. The list element is given the image and the appropriate class that styles it on the page.

```
function addMemoToPage(memo) {
  var memosUl = document.
  getElementById("memos");
  var li = document.createElement("li");
  li.innerHTML = '';
```

17. Add the text

After the image has been displayed the text is added into a paragraph element. If this is the first list element the list is added, however if there are already list elements, this text and image are added at the top so that new memos appear at the top.

```
li.innerHTML += '<p>' + memo.text + '</
p>';if (memosUl.childElementCount > 0)
{memosUl.insertBefore(li, memosUl.
firstChild);
}
else {
  memosUl.appendChild(li);
}
}
```

18. Store the memos

Now that the image and text have been added onto the screen, the code here sets about storing the memo's permanently by converting them into a JSON object. This is then stored in the local storage so that next time the app is launched the data can be retrieved and shown to the user.

```
function storeMemos() {
  var jsonMemos = JSON.stringify(memos);
  localStorage.setItem(key, jsonMemos);
}
```

19. Load the memos

When a user starts the app there needs to be a way to check if there are memos from a previous occasion and show those on the screen. This function lets that happen by checking the local storage and if there is something in

s.cordova.io/#/package/org.apache.cordova.camera

Example

Take a photo and retrieve it as a base64-encoded image:

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 50,
  destinationType: Camera.DestinationType.DATA_URL
});

function onSuccess(imageData) {
  var image = document.getElementById('myImage');
  image.src = "data:image/jpeg;base64," + imageData;
}

function onFail(message) {
  alert('Failed because: ' + message);
}
```

Take a photo and retrieve the image's file location:

```
navigator.camera.getPicture(onSuccess, onFail, { quality: 50,
  destinationType: Camera.DestinationType.FILE_URI });

function onSuccess(imageURI) {
  var image = document.getElementById('myImage');
  image.src = imageURI;
}

function onFail(message) {
  alert('Failed because: ' + message);
}
```

CameraOptions

Optional parameters to customize the camera settings.

```
{ quality : 75,
  destinationType : Camera.DestinationType.DATA_URL,
  sourceType : Camera.PictureSourceType.CAMERA,
  allowEdit : true,
  encodingType: Camera.EncodingType.JPEG,
  targetWidth: 100,
  targetHeight: 100,
  popoverOptions: CameraPopoverOptions,
  saveToPhotoAlbum: false };
```

Options

- **quality**: Quality of the saved image, expressed as a range of 0-100, where 100 is typically full resolution with no loss from file compression. The default is 50. (Number) (Note that information about the camera's resolution is unavailable.)
- **destinationType**: Choose the format of the return value. The default is FILE_URI. Defined in navigator.camera.DestinationType (Number).

```
Camera.DestinationType = {
  DATA_URL : 0,    // Return image as base64-encoded string
  FILE_URI : 1,     // Return image file URI
  NATIVE_URI : 2    // Return image native URI (e.g., assets-library:// on iOS or content:// on Android)
};
```

Cut and paste PhoneGap

If you are new to using PhoneGap it might seem like an uphill struggle knowing what the commands are and how to use them. The documentation online is excellent though and you will find many examples of code that you can simply cut and paste into your work to get it going. For example, looking at the camera API will give you code that can be dropped in. Once you have the basic functionality it simply becomes a case of building up the code until you have enough functionality for your app to work. Remember when creating apps they should have one main purpose so try and keep it to that. You could always add extra functionality once it is released and you start to receive feedback.

there, these memos are sent to the addMemoToPage function from Step 16 and 17.

```
function loadMemos() {
  var jsonMemos = localStorage.getItem(key);
  if (jsonMemos != null) {
    memos = JSON.parse(jsonMemos);
    for (var i = 0; i < memos.length; i++) {
      addMemoToPage(memos[i]);
    }
  }
}
```

20. Check local storage

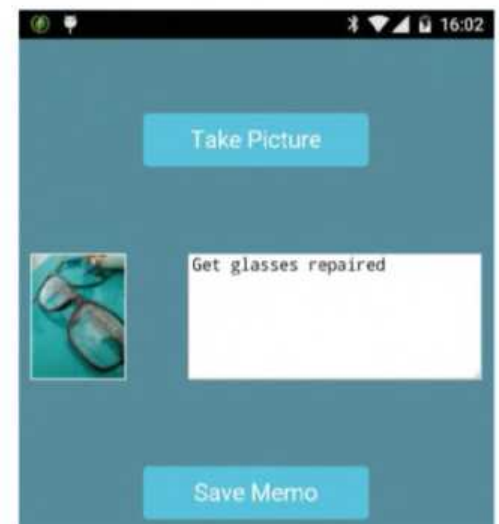
In this section of the code the 'if' statement checks if there is local storage available, if there isn't then the user is alerted to that. If there is local storage then the memos are loaded into the display by calling the

loadMemos function. Save your file now and this will automatically call the PhoneGap App on the device to reload your project.

```
if (!window.localStorage) {
  alert("The Web Storage API is not supported.");
} else {
  loadMemos();
}
```

21. Check it in action

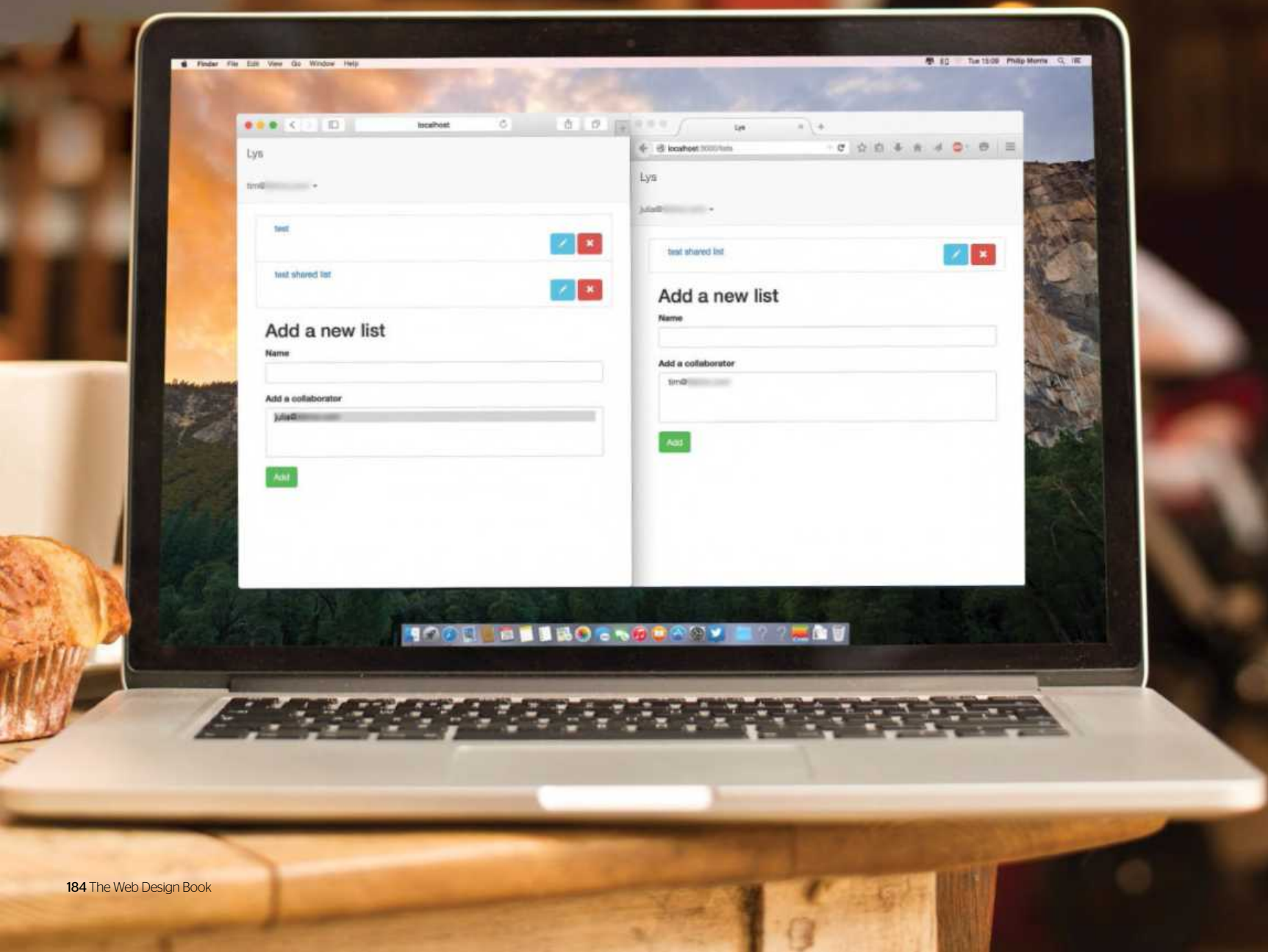
You will see a relatively empty interface but if you click the plus icon in the top-left corner of the screen, the overlay will appear and you will then be able to snap a picture and write a note. Hitting submit will save this so that next time you launch the app this data will be here ready for you.



↓ **DOWNLOAD TUTORIAL FILES**
www.filesilo.co.uk/bks-747

Develop a reactive web app with Angular-Meteor

Combine the best of both worlds with Meteor and Angular to unleash three-way binding





eteor is a great way to add 'reactiveness' to your app.

That means that as soon as a collection is updated all connected clients receive that data. Meteor uses a

library called Blaze to perform live updating from the server. Blaze requires no effort from the developer and 'just works', but by design Blaze is simpler than Angular as it has a gentler learning curve. A side effect of this is that it makes architecting a large app in Blaze challenging.

Enter Angular. It's been around for six years now and has a myriad of 'best practices'. Whether you're building a larger app or have an existing Angular app that you wish to port, Meteor and Angular are a good team. Of course, Angular isn't the only option but it's the one that we're going to focus on today.

To see how the two work together we're going to build a collaborative list app. It'll take advantage of Meteor's reactivity and Angular's structure. By the end of the tutorial you'll have a good understanding of how they complement each other.

As of version 1.0 Meteor supports Windows but it will require a different installer, aside from that the steps should be identical.

1. Install Meteor

If you haven't installed Meteor already then copy the curl command. If you're on Windows then there's a standalone installer at install.meteor.com/windows. Then create a new project with 'meteor create', this will create three files and a .meteor folder. Run it with the meteor command (this is shorthand for 'meteor run').

```
$ curl https://install.meteor.com/ | sh
$ meteor create lys
$ cd lys
$ meteor
```

2. Angular Meteor

In a single step we've already created the project and got it running via a server! The secret glue to hooking Meteor and Angular up is to use 'angular-meteor' headed by Uri Goldshtein and nine other core contributors. We can add it to the project with 'meteor add' while running and Meteor will automatically restart.

```
$ meteor add urigo:angular
```

3. Lys markup

One of the files that Meteor created for us was lys.html. This is where the head and body content of our app will reside. We'll keep it simple for now with an Angular include pointing to a file we'll create shortly. The 'base' tag is required for Angular's routing service to work in HTML5 mode.

```
<!-- lys.html -->
<head>
<title>Lys</title>
<base href="/">
</head>
<body>
<main ng-app="lys" ng-include="'index.ng.html'"></main>
</body>
```

4. Module definition

Meteor automatically loads files and because of this it has a defined load order. Files in a folder called lib are loaded before other files so this makes it a good place to put our module definition file. Note that the files within the client folder are only delivered to the client and are not run on the server.

```
// client/lib/app.js
angular.module('lys', ['angular-meteor']);
```

5. Showing a list

We'll display a list of lists on the page in a section controlled by the ListController. The 'track by' expression is important here because it means that each item won't have an internal \$\$hashKey object added to it. This is important because properties starting with \$ are protected in MongoDB and can't be added.

```
<!-- lys.html -->
<section class="list-container" ng-
controller="ListController">
<ul>
<li ng-repeat="list in lists track by
list._id"></li>
```

Best of both worlds

You can still use Blaze in your application with Angular-Meteor, more documentation can be found at angularjs.meteor.com. You can change the delimiter from '{{' to '{{' as well.

<Above>

- Mongol has a sister project called JetSetter which is a package that manages and visualises Session variables while developing

<Above>

- The buttons use icons from Twitter Bootstrap's glyphicon component aided with a title attribute

<Below>

- With a single line the collection is retrieved from the database and the work we've done so far renders it

```
</ul>
</section>
```

6. Toggle buttons

Within each list item we will now add two buttons: this will be one button to toggle editing and one button to remove it from the list. We are also using Angular's 'switch' directive to change which elements to show when we are editing. Because the input is tied to a model, updates will immediately be visible to Meteor and thus to all other users.

```
<!-- lys.html -->
<span ng-switch="edit">
<input ng-switch-when="true" ng-model="list.
name">
<p ng-switch-default>{{list.name}}</p>
</span>
<button ng-click="edit = !edit">Edit</
button>
<button ng-click="remove(list)">Remove</
button>
```

Still not convinced?

The Angular-Meteor Manifesto (angularjs.meteor.com/manifest) has a list of additional points that may convince you and also a roadmap for what's on the horizon.

7. List controller

Our list controller is pretty standard but as well as passing in \$scope we're also injecting \$meteor. This is what angular-meteor provides through the extra dependency that we added to the module which gives us Angular-ified Meteor methods. Note that we're referencing a variable called Lists, it's not in this file so where is it coming from?

```
angular.module('lys').
controller('ListController', ['$scope',
'$meteor',
function ListController ($scope, $meteor) {
$scope.lists = $meteor.collection(Lists).
subscribe('lists');
$scope.users = $meteor.collection(Meteor.
users).subscribe('users');
}
]);
```

8. List collection

Lys.js is another file that Meteor created for us when it created the project. This means that it's shared with both the client and the server. We're creating a global variable called Lists which is a database collection (a little like a table). This is what we referenced in our controller in the previous step.

```
// lys.js
Lists = new Mongo.Collection('lists');
```

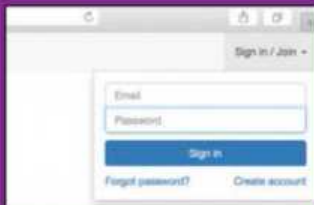
9. List permissions

By default Meteor enables anyone to edit the database from the client (via the insecure package) and the database sends everything to the client. We'll remove these later but we need to specify under what conditions a list can be inserted, updated or removed. This will be based on either being an owner or collaborator.

```
Lists.allow({
insert: function (userId, list) {
return userId && list.owner === userId;
},
update: function (userId, list) {
return userId === list.owner || list.
collaborators.indexOf(userId) > - 1;
},
remove: function (userId, list) {
return userId === list.owner || list.
collaborators.indexOf(userId) > - 1;
}
});
```

10. Publish lists

As well as setting permissions we will also need to publish 'lists' which is what the client actually subscribes to. We only want to publish the lists that the logged in user has access to though so we will need to create a query here. Queries are built up through objects and arrays. Here we will look for lists where we're either the owner or a collaborator.



<Above>

- The log-in buttons partial is very simple to use if you're happy with the default markup and styling

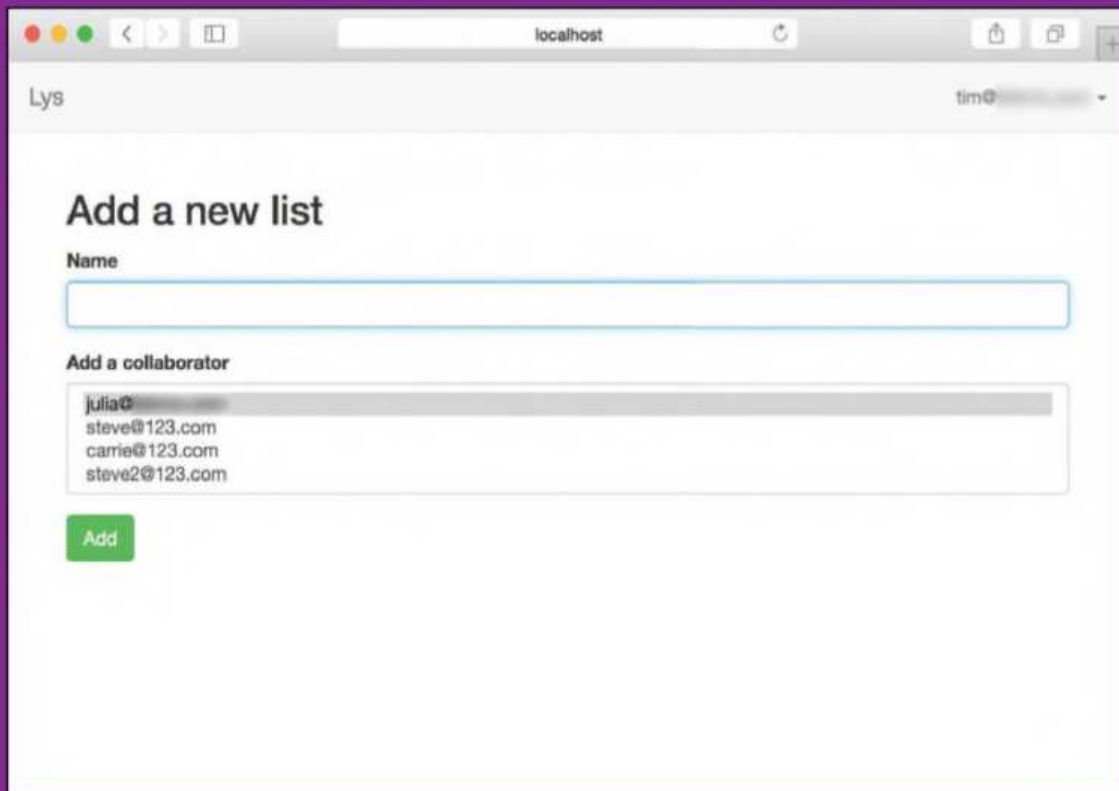


<Above>

- Although basic, the list detail view demonstrates how routing can be used in conjunction with state

<Right>

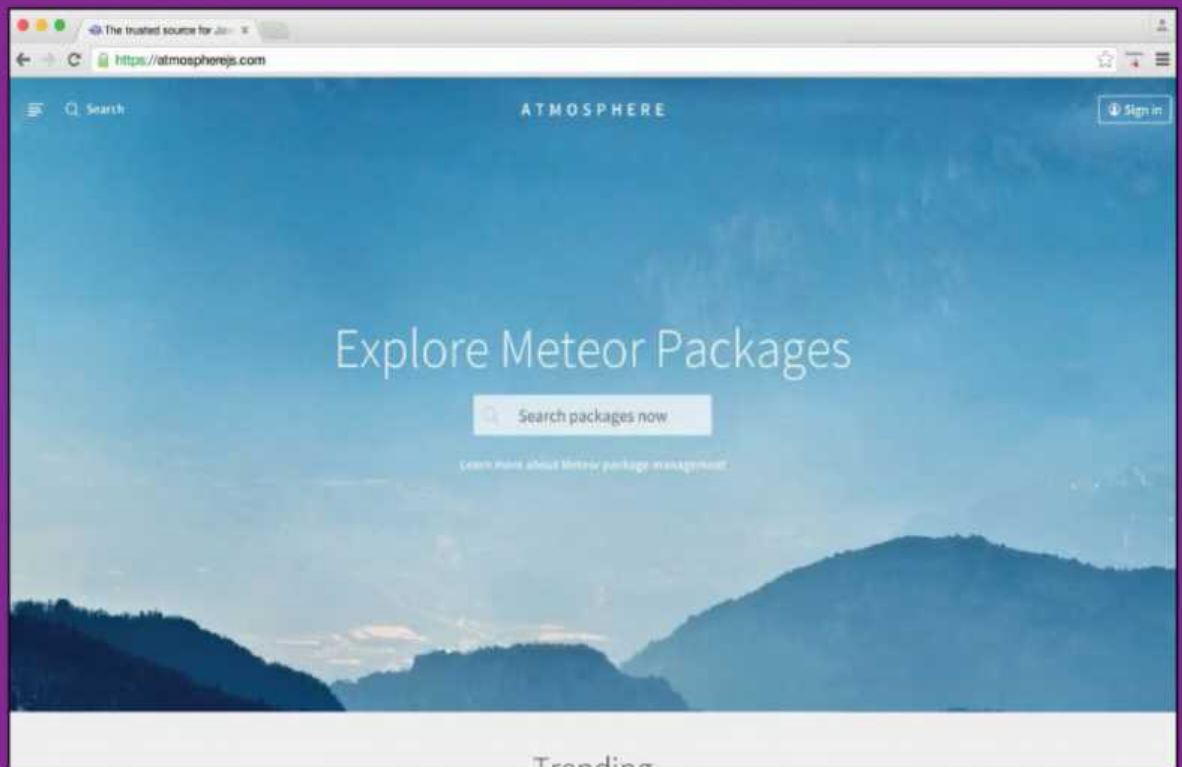
- The collaboration list is filtered to remove the current user out of the list but this could be done server-side



Where to find community code for your project

Using Angular and Meteor you have the opportunity to use packages from both communities. The de facto place for finding Meteor packages is Atmosphere (atmospherejs.com). Angular doesn't have a single place for modules but a good place to look is Angular Modules (ngmodules.org). Atmosphere is the Meteor equivalent of npm and also includes handy things like the command to paste in to add the package.

In development mode, Meteor can easily serve about 5MB of files (Firefox reckons it's closer to ten) but in production mode this is radically decreased to 240KB. This is done by running your app with the production flag, '\$ meteor --production'. Remember that each package that you add is going to increase this so ensure that you're using the package for the right reasons.



```
Meteor.publish('lists', function () {
  return Lists.find({
    $or: [{
      $and:[{
        owner: this.userId,
      }, {
        owner: {
          $exists: true
        }
      }]
    }, {
      $and:[{
        collaborators: {
          $in: [this.userId]
        }
      }]
    }
  ]);
});
```

11. Publish users

Similar to publishing lists we'll also want to publish all of our users. This is so that you can add collaborators to your lists. Note that we're returning everything (no

conditions here) but only returning the field that will be used. This means that you can't accidentally leak data that the client doesn't need.

```
Meteor.publish('users', function () {
  return Meteor.users.find({}, {
    fields: {
      emails: 1
    }
  });
});
```

12. Add or remove packages

Next we'll add basic account handling with 'accounts-password' so people can log in (yes, it's that simple!). Then add Twitter Bootstrap and a version of accounts-ui which is specific to Bootstrap 3. We'll also remove the insecure and autopublish packages as described earlier. Chaining packages by adding a space between them is a useful and quick way to act on multiple packages.

```
$ meteor add accounts-password
twbs:bootstrap ian:accounts-ui-bootstrap-3
$ meteor remove insecure autopublish
```

13. Add login buttons

Accounts-ui gives us a partial called loginButtons. This is the same as the double curly brace syntax that Angular uses but this is Meteor's Blaze template engine. It's important that the two aren't confused. The rest of the structure is for Bootstrap's navbar component.

```
<!-- lys.html --->
<nav class="navbar navbar-default">
  <div class="container-fluid">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Lys</a>
    </div>
    <div class="nav navbar-nav navbar-right">
      {{ > loginButtons }}
    </div>
  </div>
</nav>
```

14. Show form

\$root here is the same as \$scope.\$root and accesses \$rootScope without having to explicitly pass it to the template. The next steps will expand this form. \$root also has the getReactively method which watches the variable for changes and notifies Meteor's Tracker library.

```
<form class="col-md-7" ng-show="$root.
  currentUser">
  <h2>Add a new list</h2>
</form>
```

“ This is the same as the double curly brace syntax that Angular uses but this is Meteor's Blaze template engine ”

Mobile Apps

15. Name input

The form will create an object we'll novelly call 'newList'. One of the pieces of data that we'll need is the name of the list so we'll make a form text input to capture this data. Note that we don't have to specify the type because the default type for an input is text.

```
<label for="nameInput">Name</label>
<input ng-model="newList.name"
id="nameInput" class="form-control"
required>
```

16. Collaborator list

The collaborator field will let the user select multiple users (but not themselves) to add as a collaborator via email address. Angular's select directive makes this trivial but the syntax is daunting. The key part is the ng-options attribute which says, "use the email address as the text and call each item in the users array user".

```
<label for="users">Add a collaborator</label>
<select class="form-control" id="users"
name="users" ng-options="user.emails[0].
address for user in users | removeSelf"
ng-model="collaborators" multiple>
<option value="">Choose a collaborator</
```

```
option>
</select>
```

17. RemoveSelf filter

In the last step we referenced a filter called removeSelf and this filter will return all of the users that don't match the currently logged in user's ID. We are using a technique called 'duck typing' here by inferring from the fact that it has a length property - it'll be an array and also have a filter function.

```
angular.module('lys').filter('removeSelf',
function () {
return function (users) {
if (users.length) {
return users.filter(function (user) {
return user._id !== Meteor.userId();
});
} else {
return users;
}
};
});
```

18. Add button

To finish our form we'll create a button which will add newList to the Meteor collection. The code for this is a little more involved so to keep things tidy we'll call a method on the controller to deal with it rather than

write it all inline. The semicolon is optional but insert them in case we add more, it's the new 'be kind rewind'.

```
<button class="btn btn-success" ng-
click="add(newList);">Add</button>
```

19. Add function

The 'add' function maps each collaborator to just their ID so rather than include the entire object for each collaborator we only store their ID which we can look up later on. We set the list owner to the currently logged in user by accessing the root scope and we will then push this to the list. This push will also automatically update the database for us!

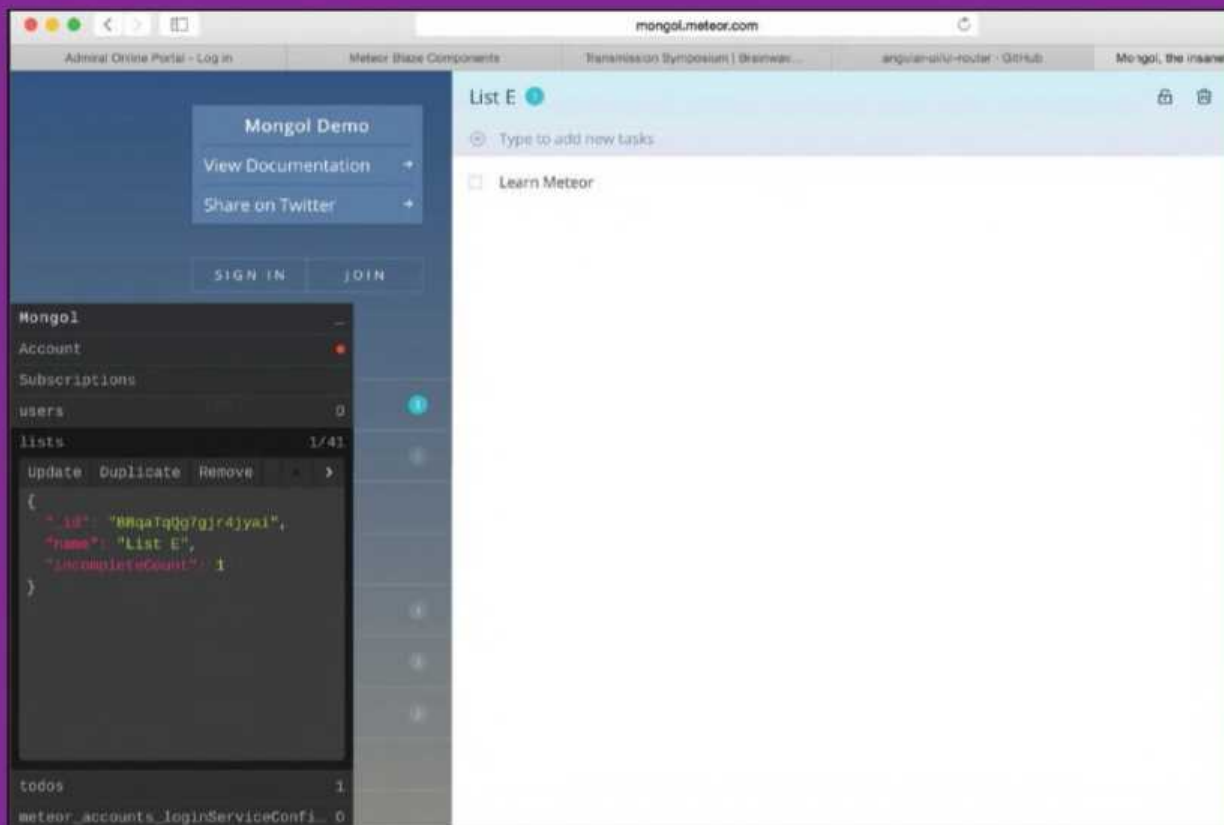
```
$scope.add = function (list) {
list.collaborators = $scope.collaborators.
map(function (user) {
return user._id;
});
list.owner = $scope.$root.currentUser._id;
$scope.lists.push(list);
};
```

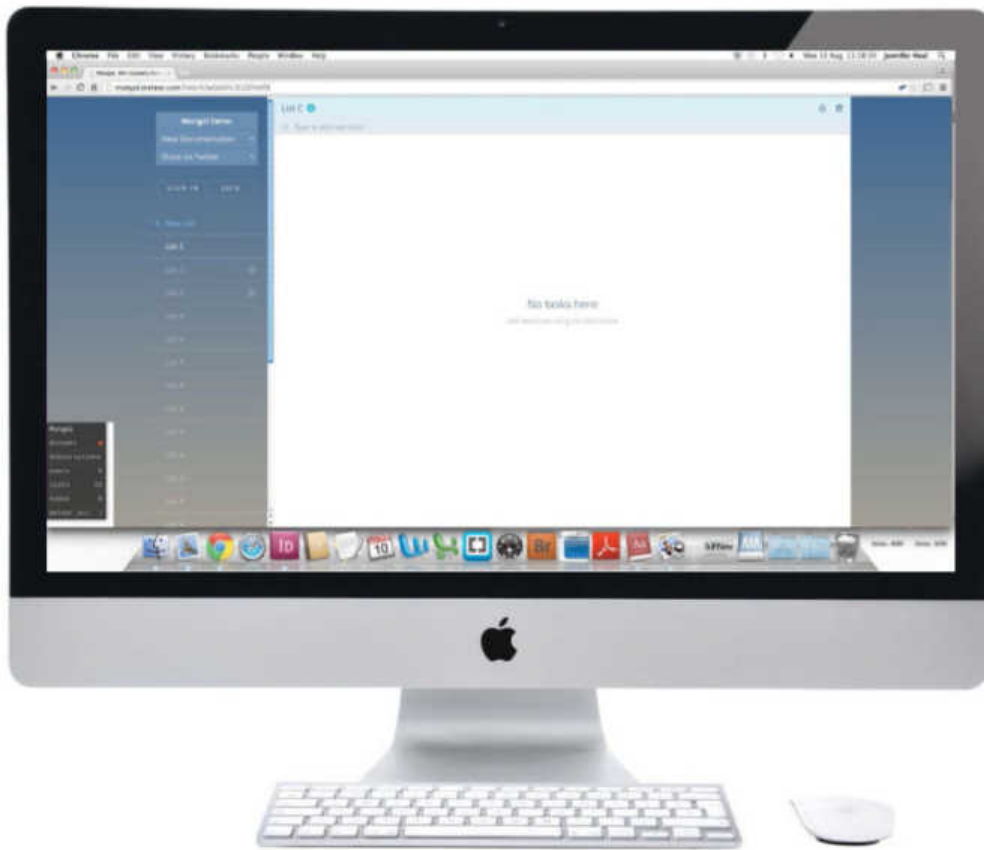
20. Remove a list

Removing a list is far simpler than adding it because all we need to do is call the remove function on the collection. This receives the list object and voila, it's

Monitor and debug with Mongol

An invaluable package while developing Meteor apps is msavin's Mongol (mongol.meteor.com). Add it to your project with '\$ meteor add msavin:mongol' and wait for Meteor to restart. You shouldn't notice anything different until you hit Cmd/Ctrl+M in the browser and a black window pops up in the bottom left. Mongol gives you real-time visibility of all the collections that are currently published to the page. This includes users! Here you can add, edit, duplicate and remove documents (analogous to records). It's useful for working out if something has made it to the database or not and can help with debugging. Mongol's smart enough to remove itself when built or deployed unless you're in debug mode.





“ As an exercise for the discerning reader, see if you can get a list of email addresses to display for each collaborator ”

removed from the database and each connected client sees the list vanish. Meteor collections have all sorts of exotic methods for you to play with like `updateCursor` and `unregisterAutoBind`.

```
$scope.remove = function(list) {
  $scope.lists.remove(list);
};
```

21. Jumbotron message

You may have noticed that we had to create a 'special' HTML file suffixed with `.ng.html`. This is because otherwise Meteor will try to parse the Angular expressions and break. Next we'll create a message for logged out users explaining the benefits of what they'll be able to do with an account. This works because `currentUser` is 'null' when there is no user.

```
<!-- index.ng.html - ->
<div class="col-md-12">
<div class="jumbotron" ng-hide="$root.currentUser">
<div class="container">
<p>Login to create and share lists with
your friends and family!</p>
</div>
</div>
</div>
```

22. Angular routing

So we can now create a list and add collaborators to it. Let's introduce routing to our little app. Create a file called 'routes.js' inside of the client folder. HTML5 mode rewrites URLs to 'normal' URLs which fallback to hashbangs in older browsers. The next steps will sit inside this function.

```
// client/routes.js
angular.module('lys').
config(['$routeProvider',
'$stateProvider', '$locationProvider',
function ($routeProvider,
$stateProvider, $locationProvider) {
$locationProvider.html5Mode(true);
$routeProvider.otherwise('/lists');
}]);
```

23. List route config

Angular's state provider comes from the ui-router package. Here we can specify what is rendered at a given URL and which controller it should be hooked up to. More information on controlling state can be found at github.com/angular-ui/ui-router/wiki.

```
$stateProvider
.state('lists', {
url: '/lists',
templateUrl: 'client/lists/views/lists.ng.html',
```

```
controller: 'ListController'
});
```

24. List detail config

Likewise we'll now be doing the same for the details view. Note that this will take a URL parameter denoted by the colon. `ListId` will be the ID of the list in the database which we will use to get further information about it. Being able to specify the template and controller separately is extremely useful, especially if we want to reuse any of the templates.

```
.state('listDetails', {
url: '/lists/:listId',
templateUrl: 'client/lists/views/list-
details.ng.html',
controller: 'DetailsController'
});
```

25. Details controller

The second controller in our app will get the list we want and a list of users. We're going to show the number of collaborators but you could extend it to allow the addition or removal of collaborators to the list. We name the `DetailsController` function (convention would be to keep it anonymous) for readable stack traces when we are debugging.

```
angular.module('lys').
controller('DetailsController', ['$scope',
'$meteor', '$stateParams',
function DetailsController ($scope, $meteor,
$stateParams) {
// next step
}
]);
```

26. Meteor reactivity

`$meteor.object` wraps a Meteor object to give it 'reactivity'. That is, the ability to update when other users' version of that object updates. Note also the second argument of 'false' to the Meteor collection. This toggles automatic client-side saving which we want to disable for 'users', by default this is true.

```
$scope.list = $meteor.object(Lists,
$stateParams.listId).subscribe('lists');
$scope.users = $meteor.collection(Meteor.
users, false).subscribe('users');
```

27. Render to page

Lastly we output the name of the list and the number of collaborators into the template. As an exercise for the discerning reader, see if you can get a list of email addresses to display for each collaborator. Harnessing Angular's principles with Meteor is a powerful way to architect your app.

```
<p>{{list.name}} has {{list.collaborators.
length}} collaborators</p>
```

Special
trial offer

Enjoyed
this book?



Exclusive offer for new



Try
3 issues
for just
£5*

* This offer entitles new UK direct debit subscribers to receive their first three issues for £5. After these issues, subscribers will then pay £25.15 every six issues. Subscribers can cancel this subscription at any time. New subscriptions will start from the next available issue. Offer code ZGGZINE must be quoted to receive this special subscriptions price. Direct debit guarantee available on request. This offer will expire 31 August 2016.

** This is an US subscription offer. The USA issue rate is based on an annual subscription price of £65 for 13 issues which is equivalent to \$102 at the time of writing compared with the newsstand price of \$14.99 for 13 issues being \$194.87. Your subscription will start from the next available issue. This offer expires 31 August 2016.



About
the
mag



**Uncover the secrets
of web design**

Practical projects

Every issue is packed with step-by-step tutorials
for HTML5, CSS3, Photoshop and more

In-depth features

Discover the latest hot topics in the industry

Join the community

Get involved. Visit the website, submit a portfolio
and follow Web Designer on Twitter

subscribers to...

**web
designer**

Try 3 issues for **£5 in the UK***
or just **\$7.85 per issue in the USA****
(saving 48% off the newsstand price)

For amazing offers please visit

www.imaginesubs.co.uk/wed

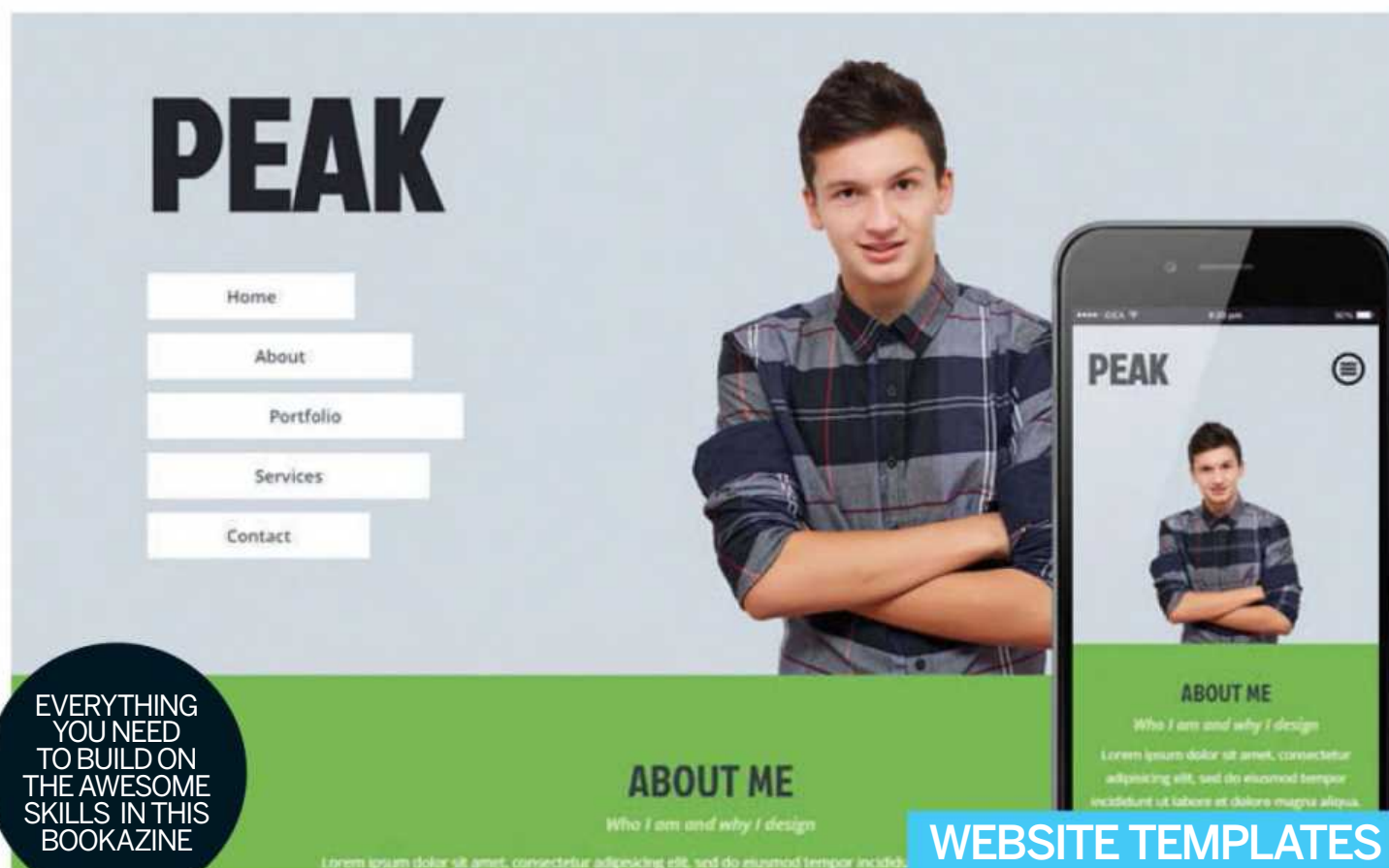
Quote code **ZGGZINE**

Or telephone **UK 0844 848 8413⁺** overseas **01795 592 878**

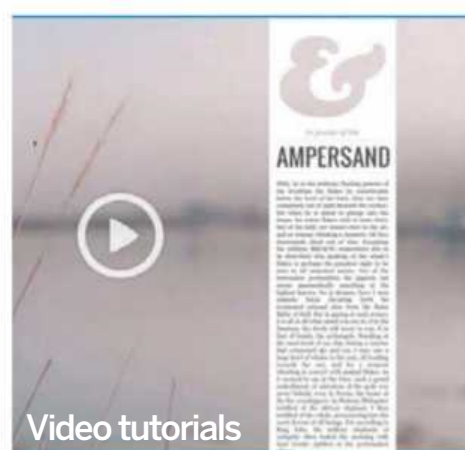
+Calls will cost 7p per minute plus your telephone company's access charge

YOUR **FREE** RESOURCES

Log in to filesilo.co.uk/bks-747 and download your great resources **NOW!**

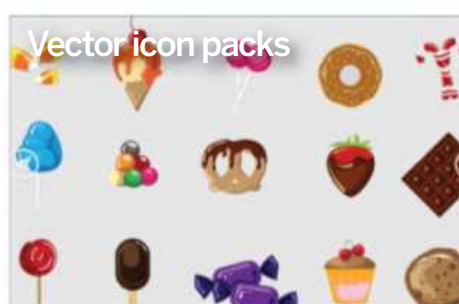


EVERYTHING
YOU NEED
TO BUILD ON
THE AWESOME
SKILLS IN THIS
BOOKAZINE



Video tutorials

PACKED WITH BRILLIANT
DIGITAL CONTENT, AVAILABLE
ANY TIME, ON DEMAND



Vector icon packs



Free font families

YOUR BONUS RESOURCES



ON FILESIL0 WITH THIS
BOOKAZINE, FREE AND
EXCLUSIVE FOR WEB DESIGN
BOOK VOLUME 5 READERS,
YOU'LL FIND A WEALTH OF
RESOURCES, INCLUDING...

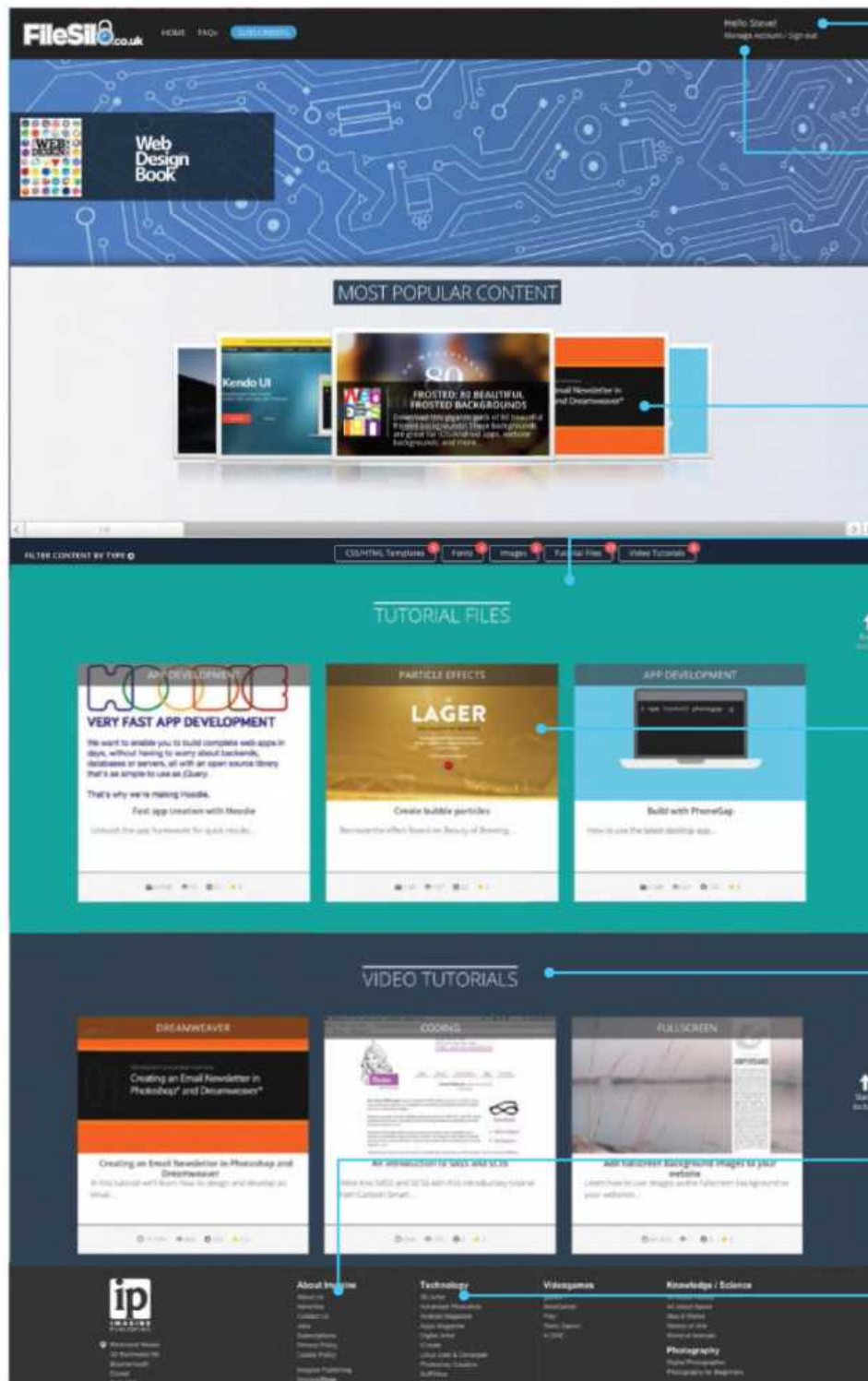
- Responsive Bootstrap templates
- Cross-browser templates
- Free font families including Manbow, Nyxali and Vexler Slip
- Over ten hours of video tuition including adding fullscreen images to your site and a guide to typography
- All the files and resources you need to complete the tutorials in the book

FileSil0

FILESILO – THE HOME OF PRO RESOURCES

Discover your free online assets

- A rapidly growing library
- Updated continually with cool resources
- Lets you keep your downloads organised
- Browse and access your content from anywhere
- No more torn disc pages to ruin your magazines
- No more broken discs
- Print subscribers get all the content
- Digital magazine owners get all the content too!
- Each issue's content is free with your magazine
- Secure online access to your free resources



This is the new FileSilo site that replaces your disc. You'll find it by visiting the link on the following page

The first time you use FileSilo, you'll need to register. After that, you can use your email address and password to log in

The most popular downloads are shown in the carousel here, so check out what your fellow readers are enjoying

If you're looking for a particular type of content, like software or video tutorials, use the filters here to refine your search

Whether it's programming tutorials or video workshops, categories make it easy to identify the content you're looking for

See key details for each resource including number of views and downloads, and the community rating

Find out more about our online stores, and useful FAQs, such as our cookie and privacy policies and contact details

Discover our fantastic sister magazines and the wealth of content and information that they provide

HOW TO USE FileSilo

EVERYTHING YOU NEED TO KNOW ABOUT ACCESSING YOUR NEW DIGITAL REPOSITORY

To access FileSilo, please visit filesilo.co.uk/bks-747

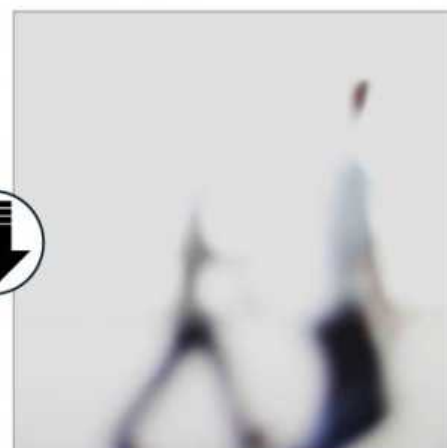
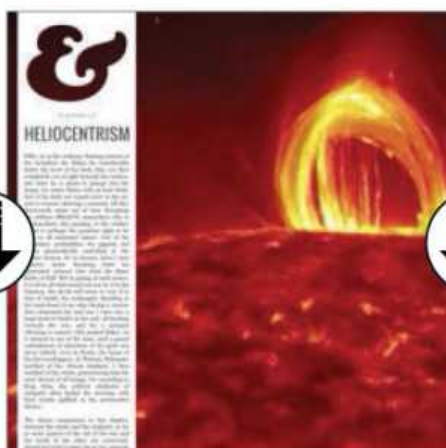
01 Follow the on-screen instructions to create an account with our secure FileSilo system, log in and unlock the bookazine by answering a simple question about it. You can now access the content for free at any time.



02 Once you have logged in, you are free to explore the wealth of content available on FileSilo, from great video tutorials and online guides to superb downloadable resources. And the more bookazines you purchase, the more your instantly accessible collection of digital content will grow.

03 You can access FileSilo on any desktop, tablet or smartphone device using any popular browser (such as Safari, Firefox or Google Chrome). However, we recommend that you use a desktop to download content, as you may not be able to download files to your phone or tablet.

04 If you have any problems with accessing content on FileSilo, or with the registration process, take a look at the FAQs online or email filesilohelp@imagine-publishing.co.uk.



NEED HELP WITH THE TUTORIALS?

Having trouble with any of the techniques in this bookazine's tutorials? Don't know how to make the best use of your free resources? Want to have your work critiqued by those in the know? Then why not visit the Web Designer and Imagine Bookazines Facebook pages for all your questions, concerns and qualms. There is a friendly community of fellow web design enthusiasts waiting to help you out, as well as regular posts and updates from the team behind Web Designer magazine. Like us today and start chatting!



facebook.com/ImagineBookazines
facebook.com/WebDesignerUK



THE WEB DESIGN BOOK

The ultimate guide to creating stunning modern websites

HTML & CSS

Power up CSS preprocessors, make pop-up boxes and much more

JQuery & JavaScript

Explore jQuery effects, add shuffling text and make games

Web Development

Find the best frameworks and discover advanced Angular.js scripts

Mobile & Web Apps

Unleash the power of the Facebook apps and build with native scripts

Free web content

Including source files, expert video tuition on HTML and CSS, web templates and more

